领域建模验证语言EDOLA研究

（申请清华大学工学博士学位论文）

培 养 单 位 ：计算机科学与技术系
学　　科 ：计算机科学与技术
研　究　生 ：张 荷 花
指 导 教 师 ：孙 家 广 教 授

二〇〇九年十一月

Domain-specific modeling and verification language EDOLA

Dissertation Submitted to

Tsinghua University

in partial fulfillment of the requirement

for the degree of

Doctor of Engineering

by

Zhang Hehua

(Computer Science and Technology)

Dissertation Supervisor : Professor Sun Jiaguang

November, 2009

关于学位论文使用授权的说明

本人完全了解清华大学有关保留、使用学位论文的规定，即：

清华大学拥有在著作权法规定范围内学位论文的使用权，其中包括：（1）已获学位的研究生必须按学校规定提交学位论文，学校可以采用影印、缩印或其他复制手段保存研究生上交的学位论文；（2）为教学和科研目的，学校可以将公开的学位论文作为资料在图书馆、资料室等场所供校内师生阅读，或在校园网上供校内师生浏览部分内容；（3）根据《中华人民共和国学位条例暂行实施办法》，向国家图书馆报送可以公开的学位论文。

本人保证遵守上述规定。

（保密的论文在解密后应遵守此规定）

作者签名：_____　　导师签名：_____

日　　期：_____　　日　　期：_____

摘 要

随着软件技术在各个行业的广泛应用，保证软件正确可靠变得越来越重要。形式验证是保证软件正确可靠的重要手段。然而，用于验证的建模语言主要是基于数理逻辑、自动机、图论等数学基础的形式语言，学习周期长且难以描述领域问题，影响了形式验证在工业界的应用。

本文研究领域建模验证语言EDOLA的设计方法和实践，以综合实现语言的领域描述易用性、复用性和自动验证性。提出了基于领域知识层、公共模块层和验证支持层的三层结构EDOLA设计方法。论文主要工作如下：

1. 在领域知识层，提出了生产调度算法和PLC控制软件两个典型领域的知识特征提取和表示方法。定义了生产调度问题、时间Petri网模型等领域知识操作符，方便了生产调度算法的描述。定义了循环扫描周期、全环境和五类验证需求等操作符，方便了PLC控制软件的描述。给出了领域操作符的形式语义，为EDOLA语言定义和自动验证提供支持。
2. 在公共模块层，以时间模块为例，提出了公共模块层的EDOLA操作符定义方法。定义了两类基本时间操作符和四种高级操作符，实现了EDOLA在时间特征描述上的易用性以及时间相关领域EDOLA语言设计的复用性。
3. 在验证支持层，提出了面向验证需求的抽象策略，减少了自动验证过程中的状态空间和搜索空间。制定了EDOLA到一阶逻辑的编码规则，从而借助一阶逻辑自动定理证明工具，实现了对无限空间模型的验证。
4. 设计了PLC领域建模验证语言EDOLA-PLC的原型并实现了相关工具。工具提供了用户界面、语言的语法语义检查和基于转换的自动验证等功能，方便了EDOLA-PLC语言的应用。
5. 给出了EDOLA-PLC在码头消防PLC控制案例上的典型应用。通过实例验证了EDOLA-PLC语言在PLC领域知识和验证需求描述上的易用性、在公共知识-时间特征描述上的易用性以及自动验证性。实验结果表明验证支持层的抽象策略提高了自动验证的有效性。

关键词：领域专用语言；TLA$^+$；自动定理证明；可编程逻辑控制器；车间调度

Abstract

With the widely use of software technique in everyday applications, the correctness of software becomes more and more important. Formal verification is an important method to improve the correctness of software. However, it mainly takes formal languages as its modeling languages, which are based on mathematical logic, automata or graph theory, hard for learning and domain description. That hinders the applications of formal verification in industry.

This dissertation investigates the design and practice of domain modeling and verification language EDOLA, to possess all the features of the usability for domain description, reusability and automatic verification. It proposes a three-level design method with the domain knowledge level, the common module level and the verification support level. The main contributions are summarized as follows:

1. In the domain knowledge level, the extraction and representation methods of the domain knowledge on both job-shop scheduling and PLC control software are proposed. It defines domain-specific operators of the job-shop scheduling problem, timed Petri net, etc. for the job-shop scheduling description. It also defines the operators of the scan cycle pattern, the complete environment pattern and five kinds of verification requests for the PLC domain description. It presents the formal semantics of the defined domain-specific operators, for the further EDOLA definition and its automatic verification.
2. In the common module level, the method to define common operators is presented with real-time as an example for common knowledge. It proposes two kinds of basic time operators and four advanced ones, which help EDOLA to describe real-time features easily and make the reusability of EDOLA design among time-sensitive domains possible.
3. In the verification support level, it presents a properties-oriented abstraction strategy, which reduces the state space and exploring space during automatic verifi-

cation. It then formulates the encoding rules from EDOLA to first-order logic, thus implements the verification of the models with infinite states, with the help of first-order logic automatic theorem provers.

4. A prototype of the PLC domain modeling and verification language: EDOLA-PLC are developed and its tools are implemented. The tools provide an EDOLA-PLC editor and a compiler with the functionalities like syntax checking, semantics checking and translation-based automatic verification.
5. A case study of the EDOLA-PLC language on a dock fire-fighting control system is presented. It indicates that EDOLA-PLC is easy to describe both the PLC domain knowledge and the properties to be verified; is easy to describe the common knowledge: real-time and can be verified automatically. The results show that the abstraction strategy adopted in the verification support level of EDOLA-PLC improves the efficiency of automatic verification.

Key words: Domain-specific language; TLA$^+$; automatic theorem proving; programmable logic controller; job-shop scheduling

目 录

主要符号对照表

EDOLA	新型领域语言 (Evolved DOmain LAnguage)
DSL	领域专用语言 (Domain Specific Language)
TLA	动作时序逻辑 (Temporal Logic of Actions)
CTL	计算树逻辑 (Computation Tree Logic)
LTL	线性时序逻辑 (Linear Temporal Logic)
SMT	可满足性模理论 (Satisfiability Modulo Theory)
PLC	可编程逻辑控制器 (Programmable Logic Controller)
UML	统一建模语言 (Unified Modeling Language)

第 1 章 绪论

1.1 研究背景

随着软件技术在各行各业的广泛应用，保证软件正确可靠变得越来越重要。尤其是对于如核电站监控、交通调度、医疗设备控制等安全攸关（Safety-Critical）或生命攸关（Life-Critical）的应用领域，软件的正确可靠显得至关重要。传统的软件测试技术不能保证测试完全，发现问题的能力有限，难以满足高可靠性要求。形式验证（Formal Verification）以数学推理或状态遍历的方法精确判定一个系统模型是否满足给定的属性要求；能揭示软件系统中可能存在的不连贯性、歧义性以及不完全性，是软件测试技术的有益补充。

形式验证一般分为模型检测（Model Checking）和定理证明（Theorem Proving）两种方法。模型检测是对有限状态系统进行自动验证的技术[1]。它通过时序逻辑或自动机描述待验证属性，通过遍历系统的状态空间判定属性是否满足。典型的模型检测工具有NuSMV[2]、SPIN[3]和UPPAAL[4]等。定理证明方法通过数学推理对模型满足属性的猜想进行证明，不受状态空间限制，理论上可处理任意规模的问题。定理证明又分为交互式和自动定理证明两类。交互式定理证明一般基于高阶逻辑理论，表达能力强，但证明过程难以自动化，需要人工指导，比较成熟的交互式定理证明工具有Coq[5]、Isabelle[6]等。自动定理证明基于命题逻辑或一阶逻辑，表达能力有限，但证明过程可以完全自动。近年来随着可满足性模理论SMT（Satisfiability Modulo Theory）技术的发展[7]，自动定理证明工具的可处理范围逐渐增大，比较有代表性的SMT自动定理证明工具有Z3[8]、CVC3[9]、SPASS[10]等。

尽管形式验证是保证软件可靠性的一种重要方法，但它对用户数学理论基础的高要求，限制了它在工业界的应用。为此，人们基于模型检测和自动定理证明方法开发了许多工具，以保证验证过程自动化。随着这两种方法及其工具的逐渐发展，形式验证的门槛逐渐降低。然而，验证工具一般需借助一种或多种规范说明语言（Specification Language）来对系统行为和验证需求进行建模，

因此，为使用形式验证工具，用户还需学习规范说明语言的语法语义，这对数学理论基础要求仍然较高，学习周期长。比如模型检测工具SMV基于计算树逻辑CTL（Computation Tree Logic）[11] 表达待验证属性，虽然CTL的语法很简洁，但其语义却难以理解和掌握。

此外，规范说明语言在描述领域知识方面，易用性较差。若我们留意程序设计语言的研究，会发现它正在经历一场革命性的变化：语言设计者的关注点从通用语言（General Purpose Language）逐渐转移到了领域专用语言DSL（Domain Specific Language）[12]。人们最初的语言设计理想是构造一个比其它任何语言都适合编程的通用语言，于是我们有了 C，C++，*C*#，Java等多种程序设计语言，而最初的理想却依然没有实现。与通用语言致力于求解任何类型的计算问题不同，领域专用语言的设计目标是便于求解某一类特定的问题，因此（对特定问题域的用户来说）它们一般比通用语言更容易学习和接受。在规范说明语言方面，事实上我们也面临着同样的问题。目前已有的规范说明语言达数十种，很难说哪一个最好。一些语言表达能力强，却不能自动验证；一些语言语法成分简单，因而较容易理解，却又不够抽象，模型描述繁冗。

为了保证形式验证在工业界的成功应用，我们期望（用于建模系统行为和验证需求的）语言能够提供用户友好的符号、概念[13]并能够完全自动验证。对于前者，领域专用语言的已有研究成果可提供帮助。然而，领域专用语言也存在它自身的问题，即语言的编译器开发难度较大。这一点可通过尽可能重用已有通用语言或其它领域专用语言的编译器，提高领域专用语言的复用性进行改善。对语言能够自动验证的要求，可避免用户陷入（最困难的）手工证明的形式验证方法中[14]。因此，本文研究领域建模验证语言（记为EDOLA ①）的设计方法，以综合实现领域知识和验证需求的描述易用性、复用性和自动验证性；从语言角度，改善形式验证在工业界的适用性。

1.2 研究现状

根据应用目的不同，诞生了多种软件建模语言。面向形式验证的建模语言一

① EDOLA是英文Evolved DOmain LAnguage的缩写，表示当前领域专用语言概念的演化。本文的研究在语言演化方面考虑了面向验证和建模语言两个新特征，但仍然沿用了缩略语EDOLA，以方便阐述。

般是通用规范说明语言，具有严格的语义。面向特定问题的领域专用语言DSL易于描述领域知识，具有专用性。本节分别对这两类语言的研究现状进行综述。

1.2.1 规范说明语言

规范说明语言通常可分为两大类：操作性（Operational）语言和描述性（Descriptive）语言[15]。操作性语言适合于描述系统从某个初始状态开始如何演进，它通常基于状态和变迁（或事件）等基本概念，以便将系统行为描述为：当某个变迁（或事件）发生时，从一个状态到下一个状态的演变。操作性语言通常基于状态变迁系统模型，比较有代表性的有时间自动机[16]、Petri网[17]、Z语言[18]、B语言[19]、抽象状态机ASM[20]、Estelle[21]等。描述性语言则适合于表达系统必须满足的静态或动态属性，一般基于逻辑或代数的形式体系。比较有代表性的有：代数模型语言CASL[22]，它描述一组类型上的操作；进程代数模型语言LOTOS[23]，它的操作是应用在基本的进程和事件上来描述事件如何发生；逻辑模型语言Coq，它本质上是由某种逻辑（如一阶逻辑、高阶逻辑等）表达的一组公式构成。此外，目前还存在一些被称为双语言（Dual-Language）的方法[24]，将操作性语言和描述性语言结合在一起使用。比较有代表性的语言如模型检测语言Promela[25]，使用有限状态机描述系统行为，使用时序逻辑表示系统需满足的属性。但此类语言表达的系统行为和属性不属于同一逻辑框架，难以开发有效的推理规则进行定理证明。

在多种规范说明语言中，TLA$^+$是基于动作时序逻辑TLA（Temporal Logic of Actions）、一阶逻辑和ZF集合论的规范说明语言[26]。它的描述分为动作层和时序逻辑层，动作层描述系统在动作发生时的演进，时序逻辑层指定系统行为必须满足的性质。TLA$^+$兼具操作性和描述性特征，能够对系统行为和待验证性质在统一的逻辑框架下描述和验证。与其它规范说明语言相比，TLA$^+$具有以下几个特点：

- 一个TLA$^+$模型本质上就是一条时序逻辑公式，完全由数学符号构成；
- 在TLA$^+$逻辑框架下，系统行为的规范说明和待验证性质可以统一描述；
- TLA$^+$提供了对具体模型“实现”抽象模型的简单数学定义：实现即蕴含；
- TLA$^+$是无类型的。模型的类型正确性表达为一个待验证的类型不变式（Invariant）；

- TLA$^+$既可以表达安全（Saftety）属性又可以表达活（Liveness）属性。

在规范说明语言及语言理论研究方面，国内比较有代表性的工作有：周巢尘院士等[27]提出了用于嵌入式实时软件系统形式化设计的时段演算（Duration Calculus）；何积丰院士等[28,29]在“设计严格安全软件的完备演算系统”以及编程统一理论（Unifying Theory of Programming）等方面取得的成果；林惠民院士等[30–32]在并发理论、消息传送进程的语义理论以及π演算的公理化等方面取得的进展；王戟教授等[33]在混合系统设计演算方面所作的工作等。

1.2.2 领域专用语言

领域专用语言DSL是面向特定问题设计的程序设计语言或建模语言，通过提供领域相关的符号和语法结构来方便描述领域问题。目前已有的DSL达数百个，以领域程序设计语言居多。它们被广泛应用于各个领域，如绘图语言FPIC[34]、用于工业自动化的动作控制语言Mocol[35]、可编程设备语言ESP[36]、多任务实时嵌入式系统编程语言Nemo[37]、高速缓存一致性协议编写语言Teapot[38]、Web计算语言Mawl[39]、路由网络配置语言Nettle[40]等。著名的数据库应用结构查询语言SQL、语法描述巴克斯范式BNF 以及网页应用超文本标记语言HTML，都是领域专用语言的典型例子。

DSL设计方法学和实现技术近年来逐渐被总结和提出。比较有代表性的工作有：Charles等[41,42]提出了基于程序族（Program Families）的自底向上研究方法以及DSL编译器的开发方法；Spinellis等[43]提出DSL的典型设计模式；Deursen等[44]和Mernik等[45]分别对 DSL的设计方法学、实现技术和典型语言实例进行了综述；Grey等[46] 对DSL的优缺点进行了探讨。

与领域程序设计语言相比，领域建模语言的研究相对较少，且主要集中在基于统一建模语言UML（Unified Modeling Language）[47]的图形化建模语言研究之上。Sztipanovits等[48]提出了模型集成计算理论MIC（Model Integrated Computing），以便为领域建模语言提供开发环境和工具。Porter等[49]开发了嵌入式系统建模语言ESMoL和相关工具。Tolvanen等[50–52]提出了领域建模语言的设计方法并进行了面向微软SmartPhone平台的语言开发实践。OMG建立了特性描述文件（Profile）机制以使UML适应于特定平台（如J2EE和.NET等）或具体领域（如医学、航空、金融等）。目前已提出的特征描述文件如：面向企业应用集

成的EAI[53]，面向实时和嵌入式系统建模分析的MARTE[54]等，但这些特征描述文件缺乏完善的语义框架，难以对其进行分析（譬如不一致性检查）。

随着形式验证方法及其工具的发展，近年来研究者们开始关注 DSL 的形式验证问题。形式验证工作主要采取基于转换的方法使用已有验证工具进行。例如，Risoldi等[55]将控制系统界面语言Cospel转换为并发面向对象Petri网，并基于Petri网的模型检测方法进行验证。 Bodeveix等[56,57]将操作系统进程调度策略描述语言Bossa转换为B模型，然后调用自动定理证明工具Mona进行验证。Latry等[58]通过将电信服务描述语言CPL转换为TLA^+模型，使用TLA^+的模型检测工具TLC[122]进行验证。若开发DSL时未定义其形式语义，则这种方法将难以保证转换后的模型与用户使用的DSL模型语义一致。此外，待验证性质仍需基于时序逻辑进行描述。

国内领域专用语言的相关研究不多。北京大学张乃孝教授等[59,60]研究了领域语言的开发方法，设计并实现了一种面向领域语言的集成开发环境Garden，并在此系统中成功开发了若干小语言如规格说明元语言GarAda[61]、控制决策语言VERT[62]等。上海交通大学的张迎春等[63]定义了一种描述软件动态更新策略的领域专用语言upDSL。华南理工大学的宋柱梅等[64]基于MIC理论开发了面向嵌入式装备控制系统的领域建模语言。北京工业大学的周艳明[65]提出了领域专用语言自动生成应用软件的设计框架，并设计了一种描述GIS应用系统的领域专用语言GASL。军械工程学院的王成等[66]设计并实现了一种面向自动测试系统的领域语言TDSL，南京大学的李英军等[67]提出了一个油气勘探领域应用软件建造和集成的模式语言SEIS++，以及哈尔滨工程大学的王爱菊[68]提出了银行信贷业务领域描述语言CBDL等。

1.3 研究思路

本文研究领域建模验证语言EDOLA的设计方法。为综合实现领域知识和验证需求的描述易用性、复用性以及自动验证性，我们提出一个三层的EDOLA设计结构，如图 1.1 所示。

EDOLA语言的三层结构自顶向下表示为：领域知识层、公共模块层和验证支持层。领域知识层，负责对领域特征的表达。另外，由于验证需求往往和领域

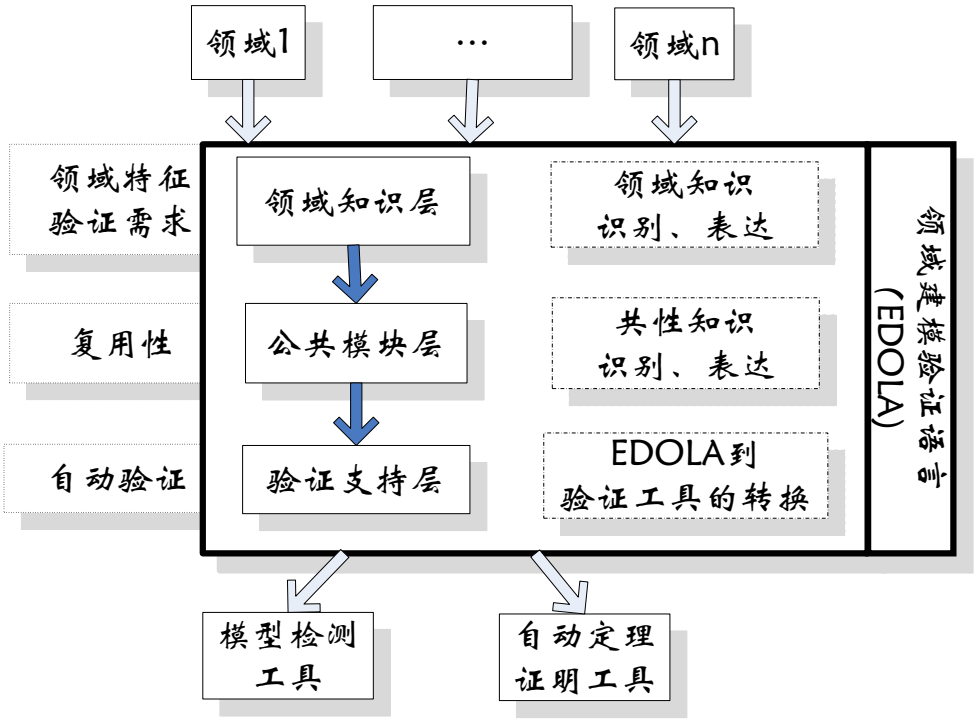

图 1.1 EDOLA的三层结构设计

问题密切相关，因此对验证需求的表达也在领域知识层实现。这一层的主要任务是识别领域知识，然后将其表达为EDOLA语言中的语法元素并为其定义形式语义。公共模块层实现EDOLA语言设计的复用性，它负责识别多个具体领域的共性知识，并将其表达为领域库中的基本操作，供领域知识层直接调用或进一步封装。验证支持层实现语言的自动验证特征，它通过实现EDOLA到模型检测和自动定理证明工具输入语言的转换，基于已有的验证工具对EDOLA模型进行自动验证。

语言的实现方法主要分为独立开发和基于基语言（Base Language）开发解释器或编译器两类。后者开发代价适中、复用性好，被广泛采用。根据第 1.2.1 节介绍，TLA$^+$是一个规范说明语言，具有良好定义的形式语义，它在抽象能力、验证需求描述和验证支持方面具有优势，在分布式算法、通讯协议、控制系统等多个领域得到了应用 [69–71]。因此，本文选择TLA$^+$作为实现EDOLA的基语言，负责解释EDOLA的形式语义以及与自动验证工具的衔接。

1.4 预备知识

本节介绍TLA逻辑的语法和语义，并对其它章节用到的部分TLA$^+$表达式进行介绍。

1.4.1 TLA逻辑的语法和语义

TLA逻辑的语法 基于TLA逻辑的规范说明本质上是一条（时序逻辑）公式，语法形式定义为

$$
\begin{aligned}
&\langle\text{公式}\rangle \triangleq \langle\text{谓词}\rangle \mid \Box[\langle\text{动作}\rangle]_{\langle\text{状态函数}\rangle} \mid \\
&\qquad \neg\langle\text{公式}\rangle \mid \langle\text{公式}\rangle \wedge \langle\text{公式}\rangle \mid \Box\langle\text{公式}\rangle \\
&\langle\text{动作}\rangle \triangleq \text{包含常量符号、变量符号和带撇变量的布尔表达式} \\
&\langle\text{谓词}\rangle \triangleq \text{不包含带撇变量的}\langle\text{动作}\rangle \mid \textsc{enabled}\ \langle\text{动作}\rangle \\
&\langle\text{状态函数}\rangle \triangleq \text{包含常量符号和变量符号的非布尔表达式}
\end{aligned}
$$

TLA逻辑的语义 TLA逻辑的语义是基于状态的概念进行定义的。一个状态表示对变量的一组赋值，所有可能状态的集合用**St**表示。非形式来讲，TLA逻辑中的动作表示旧状态和新状态之间的关系。时序逻辑的语义定义在行为之上，而行为是状态的一个无限序列。TLA逻辑的语义形式定义如下[72]。

$$
\begin{aligned}
&s[\![f]\!] \triangleq f(\forall\, \text{`}v\text{'} : s[\![v]\!]/v) && \sigma[\![F \wedge G]\!] \triangleq \sigma[\![F]\!] \wedge [\![G]\!] \\
&s[\![A]\!]t \triangleq A(\forall\, \text{`}v\text{'} : s[\![v]\!]/v, t[\![v]\!]/v') && \sigma[\![\neg F]\!] \triangleq \neg\sigma[\![F]\!] \\
&\models A \triangleq \forall s, t \in \mathbf{St} : s[\![A]\!]t && \models F \triangleq \forall \sigma \in \mathbf{St}^{\infty} : \sigma[\![F]\!] \\
&\qquad s[\![\textsc{enabled}\ A]\!] \triangleq \exists t \in \mathbf{St} : s[\![A]\!]t \\
&\qquad \langle s_0, s_1, \ldots\rangle[\![\Box F]\!] \triangleq \forall n \in Nat : \langle s_n, s_{n+1}, \ldots\rangle[\![F]\!] \\
&\qquad \langle s_0, s_1, \ldots\rangle[\![A]\!] \triangleq s_0[\![F]\!]s_1
\end{aligned}
$$

在基本符号的基础上定义的其它一些常用符号，表示如下。

$$
\begin{aligned}
&p' \triangleq p(\forall\, \text{`}v\text{'} : v'/v) && \Diamond F \triangleq \neg\Box\neg F \\
&[A]_f \triangleq A \vee (f' = f) && F \rightsquigarrow G \triangleq \Box(F \Rightarrow \Diamond G) \\
&\langle A\rangle_f \triangleq A \wedge (f' \neq f) && WF_f(A) \triangleq \Box\Diamond\langle A\rangle_f \vee \Box\Diamond\neg\textsc{enabled}\ \langle A\rangle_f \\
&\textsc{unchanged}\ f \triangleq f' = f && SF_f(A) \triangleq \Box\Diamond\langle A\rangle_f \vee \Diamond\Box\neg\textsc{enabled}\ \langle A\rangle_f
\end{aligned}
$$

其中f是一个状态函数；A是一个动作；F和G是公式； p是一个状态函数或谓词；$s, s_0, s_1, \ldots$ 等是状态；σ 是一个行为；$(\forall\, \text{`}v\text{'} : \ldots/v, \ldots, /v')$表示对变量$v$的所

有出现进行替换。

1.4.2 TLA⁺表达式介绍

TLA⁺中的表达式包括逻辑表达式、集合表达式、函数表达式、算术表达式等。大多数TLA⁺表达式都很直观，本节仅对其它章节将用到的一些特殊表达式进行介绍。

对齐的逻辑表达式 表达为“∧ < 逻辑表达式 >”和“∨ < 逻辑表达式 >”的组合形式，要求各“∧”符号对齐，各“∨”符号对齐。TLA⁺语言通过对齐和缩进的形式来取代括号，以增强可读性。此表达方法在多个合取和析取表达式嵌套的情况下尤其有效。例如

$$\begin{aligned} &\land \lor x = 1 \\ &\quad \lor x = 2 \\ &\land \lor y = 1 \\ &\quad \lor y = 2 \end{aligned}$$

与$((x = 1) \lor (x = 2)) \land ((y = 1) \lor (y = 2))$具有相同的含义。

集合幂集和广义并 集合S的幂集表达形式为 SUBSET S，例如 SUBSET $\{1, 2\} = \{\{\}, \{1\}, \{2\}, \{1, 2\}\}$；集合S的广义并表达形式为 UNION S，例如 UNION $\{\{\}, \{1\}, \{2\}, \{1, 2\}\} = \{1, 2\}$。

集合约束表达式 表达形式为$\{x \in S : P\}$，定义了这样一个集合：它是集合S的子集，且集合成员满足谓词P，例如：$\{x \in Nat : x\%2 = 0\}$ 定义了偶数集合（其中Nat表示自然数集）。

集合内涵表达式 表达形式为$\{e : x_1 \in S_1, \ldots, x_n \in S_n\}$，定义了这样一个集合：它的成员都具有$e$的形式，且$e$中的参数分别满足$x_1 \in S_1, \ldots, x_n \in S_n$，例如：$\{x + y : x \in \{1, 2\}, y \in \{3, 4\}\}$。

记录赋值表达式 表达式形式为$[elem_1 \mapsto val_1, \ldots, elem_n \mapsto val_n]$，表示对具有$n$个成员的记录进行赋值，这通过对记录的每个成员进行赋值实现。例如：$[a \mapsto x + 1, b \mapsto y]$。

1.5 论文贡献

论文的主要贡献概括如下：

1. 在领域知识层，提出了生产调度算法和PLC控制软件两个典型领域的知识特征提取和表示方法。给出了生产调度问题、时间Petri网模型的相应操作符，方便了对生产调度算法的描述；给出了循环扫描周期、全环境和五类验证需求的操作符，方便了对PLC控制软件的描述。
2. 在公共模块层，以时间模块为例，提出了公共模块层的EDOLA操作符定义方法。给出了两类基本时间操作符和四种高级操作符，保证了EDOLA在时间特征描述上的易用性和时效性领域EDOLA设计的复用性。
3. 在验证支持层，提出了面向验证需求的抽象策略，减少了自动验证过程中的状态空间和搜索空间。提出了EDOLA的自动定理证明方法，通过制定EDOLA到一阶逻辑的编码规则，借助一阶逻辑自动定理证明工具，实现了对无限空间模型的自动验证。
4. 设计了PLC领域建模验证语言EDOLA-PLC的原型并实现了相关工具。EDOLA-PLC语言的设计综合实现了三层的研究成果；EDOLA-PLC的编辑器提供了工具的用户界面；编译器提供了语言的语法、语义检查和基于转换的自动验证等功能，方便了EDOLA-PLC语言的应用。
5. 给出EDOLA-PLC在码头消防控制案例上的典型应用和相关比较。案例分析验证了EDOLA-PLC语言对PLC领域知识和验证需求描述的易用性、在公共知识-实时方面描述的易用性以及自动验证性。实验结果说明了验证支持层采用的抽象策略提高了自动验证的有效性。

1.6 论文结构

本论文共分为七章。第 2 章介绍EDOLA语言领域知识层的设计方法以及在生产调度算法和PLC控制软件两个领域的设计实践。第 3 章介绍公共模块层的

设计方法和实时公共模块的设计实践。第 4 章介绍验证支持层的设计和实现。第 5 章介绍PLC领域建模验证语言EDOLA-PLC的原型和工具实现。第 6 章介绍码头消防PLC控制系统的案例分析和相关比较。第 7 章对全文进行总结，并对下一步工作进行展望。

第 2 章 EDOLA的领域知识层研究和实现

EDOLA语言的领域知识层实现语言对领域知识和验证需求的描述易用性特征。本章以生产调度算法和PLC控制软件两个典型领域为例，介绍EDOLA语言领域知识层的设计方法和实践。研究分为领域界定、领域知识提取、语言描述和形式语义定义四个步骤。在生产调度算法实践中，分别提取了生产调度问题、时间Petri网定义、算法的五个关键步骤、顺序执行模式等领域特征以及两类验证需求，方便了生产调度算法领域的描述；在PLC控制软件实践中，分别提取了变量类别、环境模式、PLC与环境的交互模式等领域特征以及五种常见验证需求，方便了PLC控制软件领域的描述。本章的领域知识描述操作符为EDOLA的语法设计提供了基础，基于TLA$^+$定义的操作符形式语义，是自动验证的基础。

2.1 引言

面向验证的形式语言语义比较单一（基于代数、逻辑、状态变迁系统等），为了建模特定领域的问题或解决方案，需要不断地将领域中的特定概念转换为形式语言中的低层次表示，这种间接性造成了对特定问题进行形式描述的复杂性。EDOLA的设计目的之一就是要消除这种复杂性，使得面向验证的模型能够尽可能接近领域中的问题。形式验证需要行为模型和待验证属性两方面的信息。因此，EDOLA领域知识层的主要任务是研究如何通过语言的设计，将领域知识和待验证属性以贴近领域专家的形式方便地表达。

根据DSL的设计方法学，其开发过程一般分为领域分析、语言设计和语言实现三个阶段 [44]。在EDOLA三层结构设计中，领域知识层与传统DSL开发中领域分析和语言设计两阶段相对应，研究内容包括领域的识别、领域知识的收集、领域知识的语言描述和语义解释。EDOLA的领域分析研究借鉴了已有DSL设计方法学，但在语言设计部分，考虑了面向验证的特征，因此具有独特之处。

如何界定领域，目前尚无统一定义，但一般认为，开发DSL的前提条件是已有了比较成熟的领域知识。本章选取了两个典型的应用领域：生产调度算法和PLC控制软件作为研究对象来介绍EDOLA领域知识层的设计和实践。生产调

度领域的问题相对简单和规范，易于理解；文献中存在大量的生产调度算法，它们多采用自然语言或伪代码的方式描述，正确性难以保证，而开发生产调度领域的EDOLA语言可以很好地解决这些问题。可编程逻辑控制器PLC（Programmable Logic Controller）系统是一类特殊的嵌入式系统，近年来在工业界得到了广泛应用[73,74]。PLC控制软件特征鲜明，且其形式验证也是目前研究的热点[75–81]。

本章第 2.2 节介绍领域知识层的基本设计方法；第 2.3 节介绍生产调度算法领域的知识收集、表示和形式语义；第 2.4 节介绍PLC控制软件领域的领域知识层研究；第 2.5 节是对本章的小结。

2.2 基本设计方法

本节介绍领域知识收集、知识描述和语义解释的常用方法，并提出EDOLA领域知识层的设计方案。

领域知识的收集可分为三类：非形式化的领域分析、基于领域工程[82]（domain engineering）的分析和基于程序族[41]的分析。目前DSL的领域分析大多是非形式化的个案分析。基于领域工程的分析，通过系统性的领域建模方法来获取领域知识。比较著名的领域建模方法包括ODM[83]（Organizational Domain Modeling)、FODA[84]（Feature-Oriented Domain Analysis）、DSSA[85]（Domain-Specific Software Architectures）等。基于程序族的领域分析从一族相似程序中提取领域相关的共性特征，采用逆向工程技术从程序族中挖掘领域知识。FAST[86]（Family-oriented Abstractions, Specification and Translation）是基于程序族的分析方法之一。生产调度算法领域知识相对简单和规范，本章将基于FODA领域建模方法对其进行研究；而PLC控制软件领域的形式验证基础较好，本章将对其采用程序族的方法进行领域知识的收集，这里的程序族指的是PLC控制软件的形式模型族。

在实践中，从头开始设计一个全新的EDOLA语言是非常困难的。比较可行的EDOLA设计方法是基于一个已有的通用语言进行。采用这种方法，知识描述可使用基语言中提供的自定义语法进行表达（基语言须支持自定义语法机制）。本文选用的基语言TLA$^+$表达能力强，支持对新操作符的定义。一般来说，领域知识被描述为基语言的操作符，并对应基语言上的一组操作。也就是说，大部分

领域知识都可通过在通用基语言上封装一个包含领域操作符的模块（或称为领域库）表示。按照这种方法，直接使用通用语言也可以表达领域知识。但值得注意的是，并不是所有的领域知识都可表达为通用语言上的操作符，如第 2.4 节将介绍的PLC控制器的循环扫描特征，是难以通过定义 TLA$^+$的操作符直接表示的。对此类领域知识的方便表达，恰恰验证了设计领域专用语言的必要性。

由于验证需求和具体领域密切相关，本章也将其作为一类特殊的领域知识在领域知识层考虑。目前已存在一些改善验证需求描述易用性的研究。Pnueli[87]提出的线性时序逻辑LTL（Linear Temporal Logic）只包含将来操作符（Future Operators），为了更简洁方便的描述验证需求，一些研究者在原有LTL之上添加了过去操作符（Past Operators），虽未增加表达能力，但易用性更好[88]。值得一提的是Matthew等人的工作[89–91]。他们提出属性模式（Specification Pattern）的概念，以减轻基于时序逻辑编写属性时的困难。Matthew等从语义角度将属性划分为缺席模式（Absence）、存在模式（Existence）、响应模式（Response）等，并在模型检测工具Bandera中实现了对属性模式库的支持。与Matthew等人根据时序逻辑公式语义对属性进行分类不同，本章从特定领域应用来研究属性的分类和表示。

通过将领域知识封装为EDOLA语言的语法成分，领域的语义信息使用语法的形式表示出来，从而固化了领域知识，保证了领域知识的重用性。另一方面，为了形式验证，需要定义EDOLA的严格语义。TLA$^+$是一个形式语言，本节通过给出领域操作符的TLA$^+$ 解释，定义其形式语义。

2.3 生产调度算法的领域知识提取和表示

2.3.1 领域界定

考虑带批量（Batch）和准备时间（Setup Times）的作业车间（Job Shop）调度问题。经典的作业车间调度问题研究的是n个作业在m台机器上按照各自的加工顺序加工的问题，而在现实调度问题中，往往存在批量和准备时间的要求。批量表示一项工作能够一次性处理的数量。准备时间是机器在执行加工任务之前所需的一段时间。在大多数的加工车间调度研究中，准备时间常被认为可忽略或可作为加工时间的一部分进行处理。然而，在许多现实情形下，这些准备时间

不能被忽略，而且加工顺序不同，机器所需要的准备时间也不尽相同[92,93]，不能被视为加工时间的一部分。因此，本章考虑显式表达批量和准备时间的作业车间调度问题。

Petri网是一个表达能力强且灵活的图形化建模工具，能够表达现实生产调度问题中的复杂约束，因此在生产调度领域得到了广泛应用[94–98]。通过为生产调度问题建立Petri网模型，从而基于Petri网的分析方法来求解生产调度问题，是一类典型的方法。本节考虑的正是基于时间Petri网的生产调度求解方法。从而，领域被界定为：“带批量和准备时间的作业车间调度问题的时间Petri网模型构造算法”。

2.3.2　领域知识和验证需求提取

针对生产调度的时间Peri网构造算法，基于FODA方法提取领域知识的结果，由特征图 2.1所示。

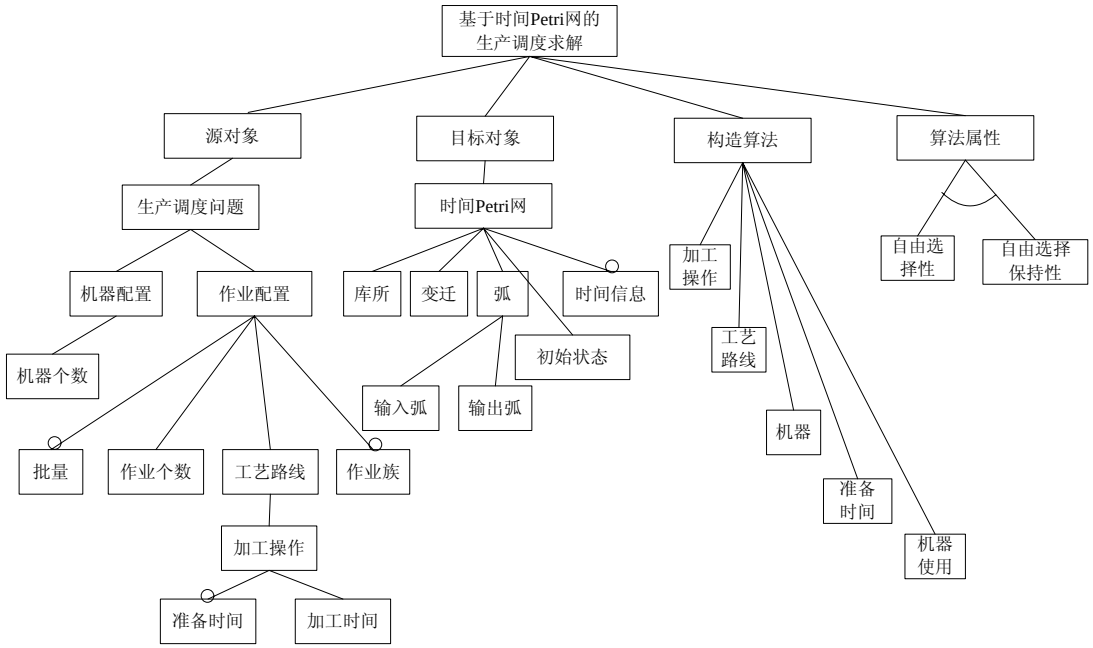

图 2.1　生产调度的时间Petri网构造算法特征图

特征图表示了特征的层次分解，其中空心圆圈表示组成父结点特征的可选

项，不带空心圆圈的部分表示组成父结点特征的必选项，而通过弧线连接起来的子结点是互斥的，即父结点只能由其中一个特征所构成。从特征图可以看出，对于生产调度问题的时间Petri网构造算法而言，领域知识包括以下几类：源对象（生产调度问题）、目标对象（时间Petri网）、构造算法和待验证性质。它们的具体描述如下。

2.3.2.1 生产调度问题

调度问题通常由三个域表示：$\alpha \mid \beta \mid \gamma$，其中 α域描述机器环境；β域指定具体的加工特征和约束条件；γ域描述优化目标[99]。假定所有的作业到达车间的时刻为0，且作业在机器之间的传输时间可忽略，则本章考察的生产调度问题表示为

α域包含：

- J_m：具有m台机器的加工车间。

β域包含：

- F：待处理的n个作业被划分为F个族（Family），$F \geq 1$。同一族中的作业，加工工艺路线相同。
- R_i：第i族作业的工艺路线，通常表示为$O_{i,1} \rightarrow \ldots \rightarrow O_{i,s_i}$，其中$s_i$为$R_i$中的操作个数。一条工艺路线不会多次使用同一台机器。
- L_i：第i族的作业个数。
- B_i：第i族作业能够被同时加工的批量。
- $ST_{k,i}$：机器k用于加工第i族的作业时所需的准备时间。对于任一台机器k，只有当它上一批处理的作业不属于第i族时，在加工第i族作业时才需要准备时间。准备时间的定义 $ST_{k,i}$被广泛应用在实际调度问题中，是更一般化的序列相关准备时间（Sequence-Dependent Setup Times）定义$ST_{k,i,j}$的特殊情形。$ST_{k,i,j}$表示机器k在处理完第i族作业，紧接着处理第j族作业时所需的准备时间。
- $P_{i,j}$：表示第i族作业的第j个加工操作的加工时间。

γ域的描述：

- 完成所有加工任务的最小流程时间（Makespan）。

2.3.2.2　时间Petri网

定义 2.1：时间Petri网是一个六元组TPN = (P, T, I, O, M_0, D)，其中P是库所（Place）的有限集合；T是变迁（Transition）的有限集合，$P \cup T \neq \varnothing$且$P \cap T = \varnothing$；$I : (P \times T) \rightarrow N$是输入函数，它定义了从库所到变迁的有向弧，其中$N$是非负整数集，整数值代表输入权值（Weight）；$O : (T \times P) \rightarrow N$是输出函数，它定义了从变迁到库所的有向弧，其中$N$中的整数值代表输出权值；$M_0 : P \rightarrow Seq$是初始状态（Initial Marking），$Seq$是由非负整数构成的序列集；$D : T \rightarrow N$是定义变迁执行时持续时间的函数。

时间Petri网的状态M，是从库所集P到Seq的一个函数，即$M : P \rightarrow Seq$，表示对时间Petri网中的库所赋予令牌（Token）值。本章考虑的时间Petri网遵循的是保持时间（Holding Durations）语义 [100]，即当变迁发生时，消耗和产生令牌的动作都瞬时发生；然而，新产生的令牌必须在它们的输出库所中保持此变迁所指定的持续时间，才能触发新的变迁。时间被加在令牌上，每一令牌都有一个时间戳，表示令牌能够被消耗的时间点。

2.3.2.3　时间Petri网模型的构造算法

在调度问题的时间Petri网构造算法中，作业车间调度问题的基本元素如作业族、批量、准备时间等被映射为时间Petri网中的基本元素。调度问题中的F个作业族被映射为F个库所，每个作业族i用一个库所f_i表示，$i = 1, \ldots, F$。向库所f_i中添加L_i个令牌来表示作业族i待加工的作业数量。批量用时间Petri网模型中有向弧上的权值来表示。准备时间用机器准备变迁上的时延表示，而处理时间则由加工变迁上的时延表示。本章研究的车间调度问题时间Petri网构造算法由五个步骤构成，分别介绍如下。下面的算法描述和图示中，对于未特殊标明的时间Petri网，规定其弧上权值为1。

加工操作建模　加工操作可以表示为三个阶段：加工前、加工和加工后。设作业族i工艺路线中的第j个加工操作为$O_{i,j}$，则$O_{i,j}$被映射为时间Petri网的变迁$pt_{i,j}$，其时延为$P_{i,j}$。此外，构造两个库所$bp_{i,j}$和$ap_{i,j}$，分别表示加工前和加工后两个阶段；加工阶段则直接由变迁$pt_{i,j}$表示。连接库所和变迁的弧上权值对应于作业族i的批量B_i，加工操作$O_{i,j}$对应的局部时间Petri网模型如图 2.2 所示。

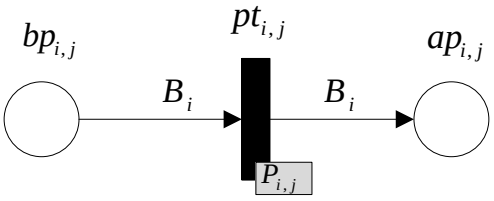

图 2.2　加工操作对应的时间Petri网模型

工艺路线建模　图 2.3 直观地展示了根据单个加工操作的时间Petri网模型，如何按照工艺路线的顺序要求，组合成工艺路线对应的局部时间Petri网模型。

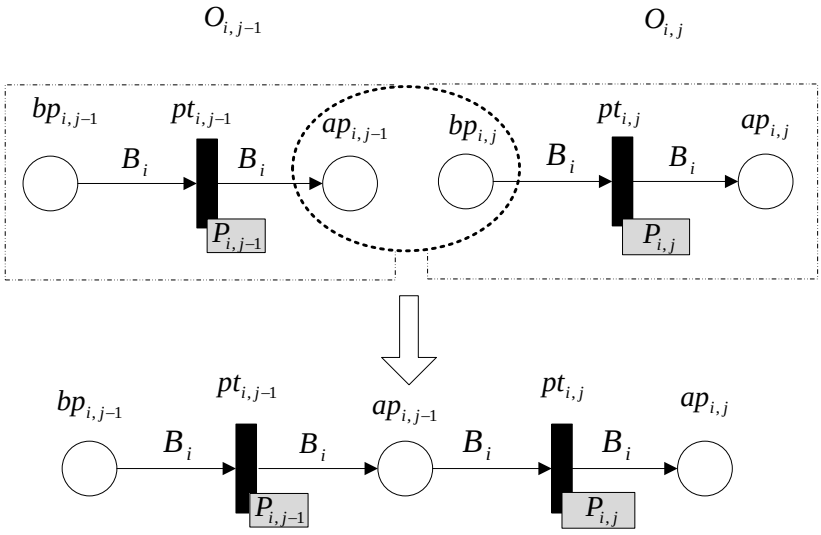

图 2.3　工艺路线对应的时间Petri网模型

假定作业族i的工艺路线R_i包含S_i个加工操作，即$R_i : O_{i,1} \rightarrow, \ldots, \rightarrow O_{i,S_i}$，则它的一个顺序关系 $O_{i,j-1} \rightarrow O_{i,j}, j = 2, \ldots, S_i$，可通过将$O_{i,j-1}$的加工后状态库所，与 $O_{i,j}$的加工前状态库所进行库所融合（Place Fusion）得到。

此外，对于工艺路线的第一个加工操作$O_{i,1}$，还需将其加工前状态库所替换为作业族i的待加工作业库所f_i，此建模过程如图 2.4 所示。

机器建模　机器的表示是构造生产调度问题的时间Petri网模型的重点，也与准备时间的建模紧密相关。在本章的建模机制中，一台机器用一个库所集合来表示。若作业车间的机器k可被c个作业族 $i_1, i_2, \ldots, i_c$使用，则它由$2 \times c + 1$个库所

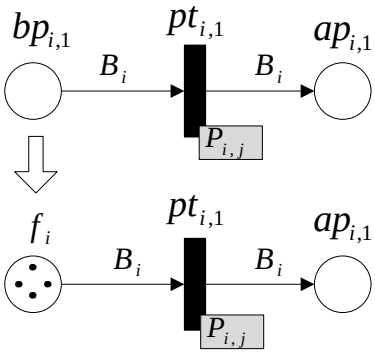

图 2.4　工艺路线的第一个加工操作对应的时间Petri网模型

构成的集合来表示：$\{kp, ka_{i_1}, ka_{i_2}, \ldots, ka_{i_c}, kb_{i_1}, kb_{i_2}, \ldots, kb_{i_c}\}$，以区分机器$k$在生产调度过程中的不同状态。机器$k$对应的局部时间Petri网模型如图 2.5 所示。

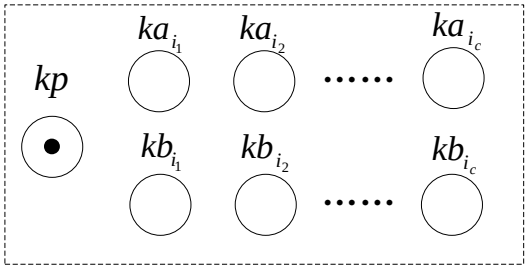

图 2.5　机器k对应的时间Petri网模型

库所kp表示机器k的初始状态，即处于空闲且尚未为任何加工操作进行准备的状态。库所ka_{i_p}表示机器k已准备好加工作业族i_p，但加工操作尚未执行，其中$p = 1, \ldots, c$。与之对应，kb_{i_p}表明机器k已准备好加工作业族i_p，且加工操作已执行，$p = 1, \ldots, c$。这里区分ka_{i_p}和kb_{i_p}的主要原因是避免机器k循环为不同的作业族进行准备，但并无任何加工操作执行的无用动作序列。

准备时间建模　根据机器的时间Petri网表示方法，准备时间的建模可通过下面三个步骤完成，如图 2.6 所示。

第一步 添加一组变迁$st0_{k,i_p}$及其连接弧，使得变迁$st0_{k,i_p}$的输入库所为kp，输出库所为ka_{i_p}，$p = 1, \ldots, c$，表示机器k从空闲状态到为作业族i_p进行准备的动作。

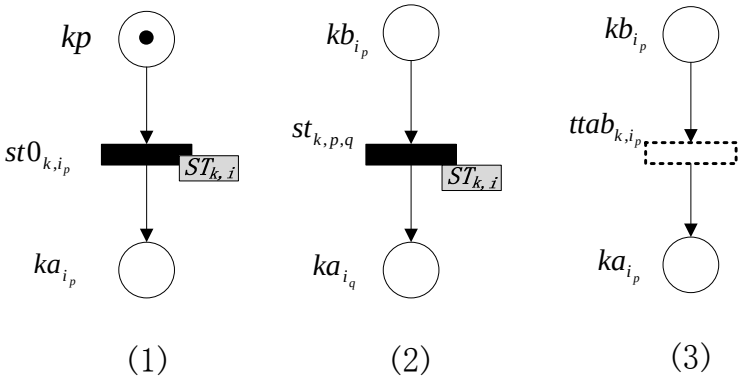

图 2.6　准备时间建模步骤对应的时间Petri网模型

第二步 对于任意两个库所kb_{i_p}和ka_{i_q}，$p, q = 1, \ldots, c, p \neq q$，添加一个变迁$st_{k,p,q}$，使得其输入库所为$kb_{i_p}$，输出库所为$ka_{i_q}$。此变迁表示机器$k$准备好并加工完作业族$i_p$之后，紧接着为另一作业族$i_q$进行准备的动作。

第三步 对于任意两个库所ka_{i_p}和kb_{i_p}，$p = 1, \ldots, c$，添加一个瞬时变迁$ttab_{k,i_p}$，其输入库所为kb_{i_p}，输出库所为ka_{i_p}。瞬时变迁是指时延为0的变迁；此局部模型表达了机器在连续加工同一作业族作业的过程中不再需要额外的准备时间这一事实。

机器使用建模　**假**定加工操作$O_{i,j}$需使用机器k，则表示加工操作的变迁$pt_{i,j}$会消耗掉库所ka_i中的一个令牌（其中ka_i中有令牌表示机器k已为此加工操作做好准备），同时在库所kb_i中生成一个新的令牌，表示加工操作已执行完毕并释放了机器k的使用权。机器使用对应的局部时间Petri网模型见图 2.7 。

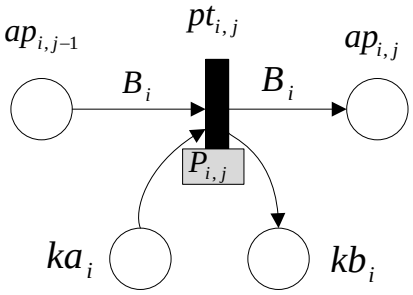

图 2.7　机器使用步骤的时间Petri网模型

2.3.2.4　待验证性质

根据Petri网结构的不同，一些比较典型的Petri网子类被提出，如状态机（State Machine）、标记图（Marked Graph）、自由选择网（Free Choice Net）、扩展自由选择网（Extended Free Choice Net）和非对称选择网（Asymmetric Choice Net）等[101]，这些Petri网子类通常具有良好的性质，使得对其进行属性验证和分析变得容易。针对生产调度问题的时间Petri网构造算法，我们希望验证得到的时间Petri网在结构上是一个自由选择网，从而和自由选择网有关的理论成果都可应用到此模型上；此外，我们也希望进一步验证构造算法的每个步骤对此自由选择性质的保持性。这两条性质和相关定义介绍如下。

定义 2.2：对于时间Petri网TPN = (P, T, I, O, M_0, D)，我们定义${}^\bullet t = \{p \mid I(p,t) \geq 1\}$为变迁$t$的输入库所集合。$t^\bullet = \{p \mid O(p,t) \geq 1\}$为变迁$t$的输出库所集合。对于库所也有相应的概念，即${}^\bullet p = \{t \mid O(p,t) \geq 1\}$为库所$p$的输入变迁集合，而$p^\bullet = \{t|I(p,t) \geq 1\}$表示库所$p$的输出变迁集合。这个概念也可以扩展到集合上，即对于一个集合$S \subseteq P$，${}^\bullet S$是S中所有成员x对应集合${}^\bullet x$的广义并，$S^\bullet$是S中所有成员x对应集合$x^\bullet$的广义并，$x \in S$。

定义 2.3：时间Petri网在结构上被称为是一个自由选择网，当且仅当$\forall p \in P$：$|p^\bullet| \leq 1$ 或 ${}^\bullet(p^\bullet) = \{p\}$。

性质 1（自由选择性质）生产调度的模型构造算法最终得到的时间Petri网是一个自由选择网。

性质2（自由选择性质的保持）生产调度的模型构造算法，保证了它的每一步骤执行之后所得局部时间Petri网，都是一个自由选择网。

2.3.3　领域知识和验证需求表示

如何根据领域分析阶段获取的领域知识，设计EDOLA中的领域操作符，并没有一般的原则。领域知识封装的粒度取决于具体的需求。封装粒度越大，得到的EDOLA抽象层次越高，易于用户操作，但不易于理解和排错。反之，EDOLA的抽象层次越低，用户使用难度越大，但易于重用。考虑极端情况，当抽象层次最

高时，整个领域问题被抽象为一个操作符；当抽象层次最低时，EDOLA 退化成了一个通用语言。

例如，可将“时间Petri网”作为一个知识来描述，将其封装为一个抽象数据类型TPN，从而在EDOLA中使用$pn \in TPN$即可表达一个时间Petri网。但也可从较低层次进行考虑，将其表示为四元组，由局部状态集Ps，变迁集Ts，输入弧集Is和输出弧集Os构成。为了便于阐述领域知识的操作符及其形式语义，本章选择在较低的抽象层次表达领域知识。下面分别对我们选取的领域知识定义其操作符。本章使用两类操作符：常量操作符和变量操作符来满足不同的领域知识描述需求。

2.3.3.1 生产调度问题

生产调度问题作为算法的输入，在算法运行过程中保持不变，因此使用常量操作符来对其进行表示。我们将生产调度问题的每个组成元素视为一个领域知识，分别表达为常量操作符 JM、$MN(\_)$、F、$O(\_)$、$Sigma(\_,\_)$、$P(\_,\_)$、$L(\_)$、$B(_)$和 $ST(\_,\_)$。

JM是车间所有机器的集合；对于所有的$m \in JM$，$MN(m) \in Nat$，表示车间中机器m的数量（Nat是TLA$^+$中的自然数集合表示）。F是作业族的集合。工艺路线通过两个参数$O(_)$和$Sigma(_,_)$共同表示：对于所有的$i \in F$，$O(i)$的值为作业族i的工艺路线上的操作个数，$O(i) \in Nat$；对于所有的$i \in F$和$1 \leq j \leq O(i)$，$Sigma(i,j)$ 的值为作业族i的工艺路线的第j个操作要使用的机器，即$Sigma(i,j) \in JM$。$P(i,j)$的值为加工时间，$P(i,j) \in Nat$。对于所有的$i \in F$，$L(i)$的值为作业族i的作业数目，$L(i) \in Nat$。$B(i)$的值为作业族i的加工批量，$B(i) \in Nat$。对于所有的$m \in JM$和$i \in F$，$ST(m,i)$的值为机器m为加工作业族i中的作业所需要的准备时间，$ST(m,i) \in Nat$。

2.3.3.2 时间Petri网

如前所述，时间Petri网由4元组$\langle Ps, Ts, Is, Os\rangle$表示，四个成员被视为四个抽象数据类型。

Ps是局部状态的集合，每个局部状态由一个包含$name$和$token$域的记录表示。这两个域分别记录库所的名字和库所中的令牌分布。为了完全形式化的

描述，还需给出记录的每个域对应的数学对象。名字一般采用字符串表示，但为解析方便，我们使用列表来表示名字。列表的相关操作在TLA$^+$ 的标准模块Sequence中定义[26]。令牌的分布包含令牌个数和令牌时间戳两方面的信息，可采用多集（Multiset）表示，但为模型构造过程描述方便，我们选择有序列表来表示$token$域。

Ts是变迁的集合，每个变迁由一个记录表示，包含$name$和$delay$两个域，分别记录变迁的名字和时延。

Is是输入弧的集合，每条输入弧由一个包含三个域的记录表示，其中$place$，$transition$和$weight$三个域分别表示变迁的输入库所名、变迁名和输入弧上的权值。

Os是输出弧的集合，每条输出弧也由一个包含$place$，$transition$和$weight$域的记录表示，这三个域分别表示变迁的输出库所名、变迁名和输出弧上的权值。

2.3.3.3 时间Petri网模型的构造算法

本章将时间Petri网构造算法的领域知识分别表示为5个建模步骤和顺序执行特征。这5个建模步骤分别对时间Petri网四元组成员*Ps*、*Ts*、*Is*和*Os*进行相应的修改，按照建模步骤顺序分别表示为：*OperationModeling(Ps, Ts, Is, Os)*、*RouteModeling(Ps, Ts, Is, Os)*、*MachineModeling(Ps, Ts, Is, Os)*、*SetupModeling(Ps, Ts, Is, Os)* 和 *MachineUseModeling(Ps, Ts, Is, Os)*。顺序执行特征不能简单地通过一个领域操作符表达，因此在EDOLA语言中可用一个关键字如**SEQ** 标识动作的顺序发生，并在编译时根据此关键字进行相应的翻译。

2.3.3.4 待验证性质

我们将两个待验证性质，分别使用操作符 *FreeChoiceProp* 和 *KeepFreeChoiceProp* 表示。

2.3.4 操作符的形式语义

本节基于形式语言TLA$^+$给出各领域知识操作符的语义解释。生产调度问题和时间Petri网的领域操作符，直接对应TLA$^+$中的常量操作符和变量定义，无需

解释。下面分别对构造算法和待验证性质的对应元素给出其TLA⁺解释。

2.3.4.1　时间Petri网的构造算法

时间Petri网构造算法的五个步骤所对应的领域操作符，分别被解释为TLA⁺中的五个动作定义。为了简洁，这里仅列出*OperationModeling*操作符和*RouteModeling*操作符的语义解释，其它三个操作符的解释可通过类似的方法给出。

加工建模操作符$OperationModeling$被解释为TLA⁺的动作定义

$$
\begin{aligned}
&OperationModeling(Ps, Ts, Is, Os) \triangleq \\
&\land\ Ts' = Ts \cup \text{UNION } \{\{[name \mapsto pt(i,j), delay \mapsto P(i,j)] \\
&\qquad\qquad : j \in 1..O(i)\} : i \in F\} \\
&\land\ Ps' = Ps \cup \text{UNION } \{\{[name \mapsto bp(i,j), token \mapsto \langle\rangle] : j \in 1..O(i)\} : i \in F\} \\
&\qquad\quad \cup \text{UNION } \{\{[name \mapsto ap(i,j), token \mapsto \langle\rangle] : j \in 1..O(i)\} : i \in F\} \\
&\land\ Is' = Is \cup \text{UNION } \{\{[place \mapsto bp(i,j), trans \mapsto pt(i,j), weight \mapsto B(i)] \\
&\qquad\qquad : j \in 1..O(i)\} : i \in F\} \\
&\land\ Os' = Os \cup \text{UNION } \{\{[place \mapsto ap(i,j), trans \mapsto pt(i,j), weight \mapsto B(i)] \\
&\qquad\qquad : j \in 1..O(i)\} : i \in F\}
\end{aligned}
$$

其中，函数$pt(i,j)$，$bp(i,j)$和$ap(i,j)$分别对应图 2.2 中的$pt_{i,j}$，$bp_{i,j}$和$ap_{i,j}$。这三个函数是表示变迁或库所名的有序列表，定义为（$\circ$是列表连接符）

$$
\begin{aligned}
pt(i,j) &\triangleq \langle\text{“pt\_”}\rangle \circ \langle i\rangle \circ \langle j\rangle \\
bp(i,j) &\triangleq \langle\text{“bp\_”}\rangle \circ \langle i\rangle \circ \langle j\rangle \\
ap(i,j) &\triangleq \langle\text{“ap\_”}\rangle \circ \langle i\rangle \circ \langle j\rangle
\end{aligned}
$$

动作$OperationModeling$表示，对于每一个作业族（$i \in F$）的每一个操作（$j \in 1..O(i)$），向变迁集合Ts中添加名字域为$pt(i,j)$且时延域为$P(i,j)$的一条记录；向局部状态集合Ps添加两条记录：一条记录的名字域为$bp(i,j)$，令牌域为空列表，另一条记录的名字域为$ap(i,j)$，令牌域也为空列表；向输入弧集合Is添加一条库所域为$bp(i,j)$、变迁域为$pt(i,j)$且权值域为 $B(i)$的记录；向输出弧集合添加一条库所域为$ap(i,j)$、变迁域为$pt(i,j)$且权值域为$B(i)$的记录。这里

运用了TLA$^+$语言中的集合内涵和集合广义并（UNION）表达式，以从既有集合中按照规则构造新的集合。以构造Ts'时∪的右部集合为例，这个UNION 集合表示的含义为

$$
\begin{aligned}
& t \in \text{UNION } \{\{[name \mapsto pt(i,j), delay \mapsto P(i,j)] : j \in 1..O(i)\} : i \in F\} \\
\equiv & \exists i \in F : t \in \{[name \mapsto pt(i,j), delay \mapsto P(i,j)] : j \in 1..O(i)\} \\
\equiv & \exists i \in F : (\exists j \in 1..O(i) : t = [name \mapsto pt(i,j), delay \mapsto P(i,j)])
\end{aligned}
$$

可以看出，此 UNION 集合所包含的成员就是我们期望添加的所有变迁。

工艺路线建模操作符$RouteModeling$被解释为TLA$^+$中的动作定义

$$
\begin{aligned}
& RouteModeling(Ps, Ts, Is, Os) \triangleq \\
& \wedge Is' = \{ \text{LET } i \triangleq x.trans[2] \quad j \triangleq x.trans[3] \text{ IN} \\
& \qquad [place \mapsto \text{IF } j > 0 \text{ THEN } ap(i, j-1) \text{ ELSE } f(i), \\
& \qquad trans \mapsto x.trans, \\
& \qquad weight \mapsto x.weight] : x \in Is\} \\
& \wedge Ps' = \{x \in Ps : Head(x.name) \neq \text{“bp\_”}\} \\
& \qquad \cup \{[name \mapsto f(i), token \mapsto Insert(\langle\rangle, L(i), 0)] : i \in F\} \\
& \wedge \text{UNCHANGED } \langle Ts, Os \rangle
\end{aligned}
$$

从图 2.3 和图 2.4 中可以看出，工艺路线建模步骤只影响到输入弧集合Is和局部状态集合Ps，而对变迁集合 Ts和输出弧集合Os没有影响，这一点由UNCHANGED 语句表达。对输入弧集合的影响由Is'表示：对于每一条输入弧x，设其变迁域表示的是作业族i的第j个操作（由名字$pt(i,j)$表示），则若是第1个操作，则将其库所域替换为$f(i)$，否则替换为$ap(i, j-1)$，其它域保持不变。从输入弧的变迁域，可以解析出当前操作所在的作业族i和加工操作编号j，在 LET IN 表达式中作为局部定义给出。对局部状态集合的影响由Ps'表达，使用了集合约束和集合内涵表达式，表示删除所有以“bp_”开头的成员，并对于每一作业族$i \in F$，添加一个记录，其名字域为$f(i)$，令牌域为$L(i)$个0构成的有序列表（表示$L(i)$个时间戳都为零的令牌）。此列表的构成调用了函数$Insert(s, m, val)$，向一个有序列表s中添加m个值为val的元素，得到一个新的有序列表。

2.3.4.2　顺序执行模式

算法的顺序执行模式，不能由TLA⁺中的自定义操作符表示。通过引入一个辅助变量 $step$表示算法步骤，算法5个步骤的顺序执行解释为下面 $\triangleq$ 右部的TLA⁺表达式，从而也可通过动作定义将构造算法表示为一个TLA⁺动作$Next$：

$$\begin{aligned} Next \triangleq & \vee (step = 0 \wedge OperationModeling(Ps, Ts, Is, Os) \wedge step' = 1) \\ & \vee (step = 1 \wedge RouteModeling(Ps, Ts, Is, Os) \wedge step' = 2) \\ & \vee (step = 2 \wedge MachineModeling(Ps, Ts, Is, Os) \wedge step' = 3) \\ & \vee (step = 3 \wedge SetupModeling(Ps, Ts, Is, Os) \wedge step' = 4) \\ & \vee (step = 4 \wedge MachineUseModeling(Ps, Ts, Is, Os) \wedge step' = 5) \end{aligned}$$

2.3.4.3　待验证性质

待验证性质$FreeChoiceProp$被表示为TLA⁺中的一个不变式

$$FreeChoiceProp \triangleq \Box(step = 5 \Rightarrow FreeChoiceDef)$$

这个定义使用了顺序执行模式解释中引入的辅助变量$step$以及公式蕴含来描述算法结束时要满足的自由选择性质验证需求。而$KeepFreeChoiceProp$被解释为

$$KeepFreeChoiceProp \triangleq \Box FreeChoiceDef$$

则表示时间Petri网在模型构造过程中始终保持自由选择性质，因此不包含步骤变量和公式蕴含。其中，$FreeChoiceDef$是自由选择网的TLA⁺定义，表示为

$$\begin{aligned} FreeChoiceDef \triangleq \forall p \in Ps : \vee\ & Cardinality(PostSet(p)) \leq 1 \\ \vee\ & PreSet(PostSet(p)) = \{p\} \end{aligned}$$

其中$Cardinality(S)$返回一个有限集合的势，它在TLA⁺的标准模块$FiniteSet$中定义。$PostSet(p)$定义了$p^\bullet$表示的集合；$PreSet(S)$定义了$^\bullet S$表示的集合。

2.4　PLC控制软件的领域知识提取和表示

本节介绍面向PLC控制软件领域的知识提取和表示。与生产调度算法领域的研究类似，我们也分别从领域界定、领域知识提取、表示和形式语义四个方面分别介绍。

2.4.1　领域界定

PLC是为取代传统继电器控制系统而设计的新型工业控制器，它采用可编程的存储器，以便在其内部存储和执行逻辑运算、顺序运算、计时、计数以及算术运算等操作指令，并通过数字或模拟的输入输出，来控制各种类型的机械或生产过程[102,103]。PLC领域的研究涉及到多个方面：PLC硬件、系统软件和用户软件等。由于PLC硬件和系统软件由PLC生产厂商所提供，因此目前大多数研究集中于PLC用户软件范畴。另一方面，PLC用户程序要根据生产厂商提供的特定编程语言编写，而不同厂商（如西门子公司的SIMATIC S7系列和Rockwell公司的ControlLogix 5550系列）甚至同一厂商的不同版本之间（如西门子公司的S7 200, 300和400系列），编程语言差别较大，这使得直接面向用户程序的研究难以具有通用性。综合以上分析，本节将研究领域界定为“PLC控制应用软件的设计”。此外，我们目前仅考虑包含数字输入输出的应用软件，从而省去了对模拟量和数字量转换的考虑。

2.4.2　领域知识和验证需求提取

建模和验证PLC控制软件要考虑的关键知识，反应了控制类软件以及基于PLC的实现不同于其它方法的特点。在PLC领域知识方面，我们介绍PLC的变量类别、环境描述模式、PLC控制软件和环境的交互模式等。此外，根据PLC控制软件的应用实践及已有形式验证研究中涉及的验证需求等，我们总结出了五类常见的验证需求。

2.4.2.1　变量类别

在程序设计语言中，变量通常根据其类型被划分为基本类型变量和组合类型变量两类。考虑领域特征时，PLC以循环扫描的方式工作，而一个PLC扫描周期由输入采样阶段、应用程序执行阶段和输出刷新阶段构成。根据这一特征，可自然地将变量划分为：输入变量、输出变量和系统变量三类，其中输入变量与PLC的输入状态寄存器对应；输出变量与PLC的输出状态寄存器对应；系统变量对应于PLC的内存变量。

2.4.2.2　环境描述

PLC控制软件通过不断地与外界环境交互，接收环境输入并输出控制信号到环境中。PLC控制软件的环境包括控制面板上的用户输入命令以及被控设备的反馈信号等，它们是 PLC控制器的输入，同时又和PLC的输出信号密切相关。基于实践，PLC的环境描述可总结为两种方式。一种比较直观的要求是在用户正常操作的前提下，PLC系统应完成所要求的功能。另一种情况是在高可信要求下，PLC控制软件必须考虑所有可能的环境输入，以便保证即使在恶意攻击、设备崩溃等特殊情况下，系统也不会崩溃或出现不可估计的错误。这种情况下，要求建立环境的“全模型”，包括用户输入和设备反馈的所有可能组合。

2.4.2.3　PLC控制软件和环境的交互模式

PLC控制系统属于典型的反应式系统。反应式系统与其环境的交互方式通常以某种方式的同步来表示。同步的实现方式有多种，下面介绍常见的三种同步实现模式。为便于理解，下面以控制软件和被控电机的一个简单交互为例，借助自动机的图形表示介绍这三种建模方式。控制软件通过发送$start$命令启动电机，并在接收到电机运动一圈的$finish$信号后回到初始状态。电机则在接收到$start$命令后，开始运转，运动一圈后，发送$finish$信号并返回到初始状态。

同步信号实现的对称同步模式　通过语言中的同步信号机制实现的控制软件与环境（电机）的同步模式如图 2.8 所示。其中 $finish?$ 和 $finish!$ 表示两个要同

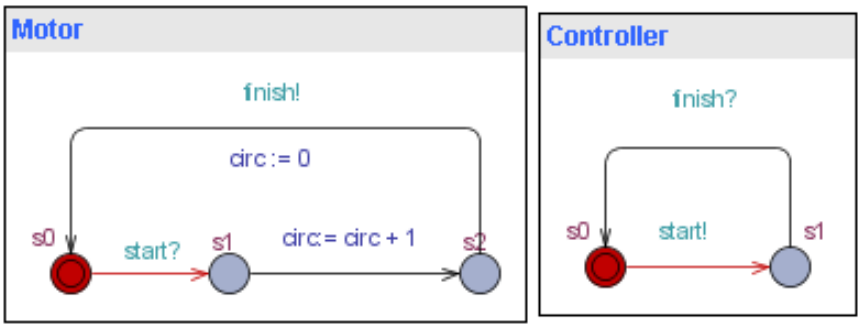

图 2.8　同步信号实现的对称同步模式示意图

步的信号，一般将*finish*! 理解为信号发送，将 *finish*? 理解为信号接收。*start*? 和 *start*! 是另一对要同步的信号。这是描述反应式系统与其环境交互的最常用模式，发送方和接收方在交互过程中是对等的，也就是说，一对同步信号的发送和接收是同时执行的。

异步执行组件的非对称同步模式　此同步模式如图 2.9 所示，环境和控制器交替执行。发送方具有较高的优先级（不必等待接收方，如 $start := true$），而接收方则使用条件判断（如 $start == true$）来等待发送信号的到达，当条件满足时执行对应的动作，并同时将对应的条件设置为假（如 $start := false$），完成一次交互。

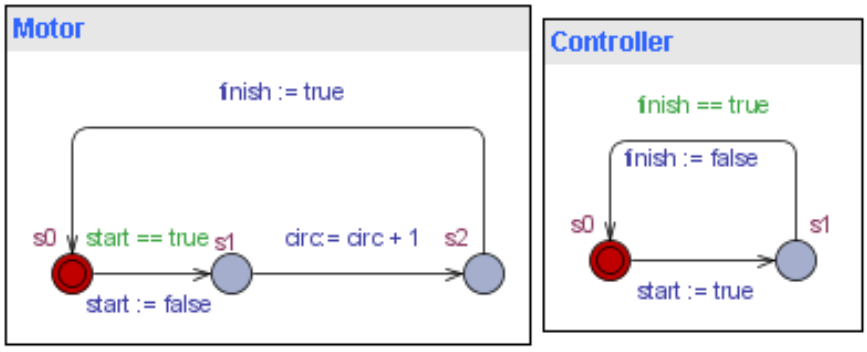

图 2.9　异步执行组件的非对称同步模式示意图

异步执行组件的外部协调的对称执行模式　此同步模式通过外部的一个协调变量（或信号）来将环境和控制器的交替执行组织起来，如图 2.10 所示，组件 *Coordinator* 用于实现环境和控制器的交替执行。协调组件发送 *env*! 信号时，电机模型执行一步；发送 *control*! 信号时，控制器模型执行一步。

根据不同的应用需求，在建模时会分别选取不同的同步模式。第一种同步模式要求语言中提供同步信号，基于此模式建立的模型简洁，但描述的抽象层次较高，和最终实现相差较远。而且，此模式与PLC控制器的实际运行差别较大。譬如对于此电机控制实例，PLC控制器根据控制器内部计算，发送启动电机的*start*命令，并不需要等待电机模型中的计算情况。实际的PLC实现中，发送方和接收方是不对等的，只有接收方需要等待发送信号到达，而发送方不需等待。

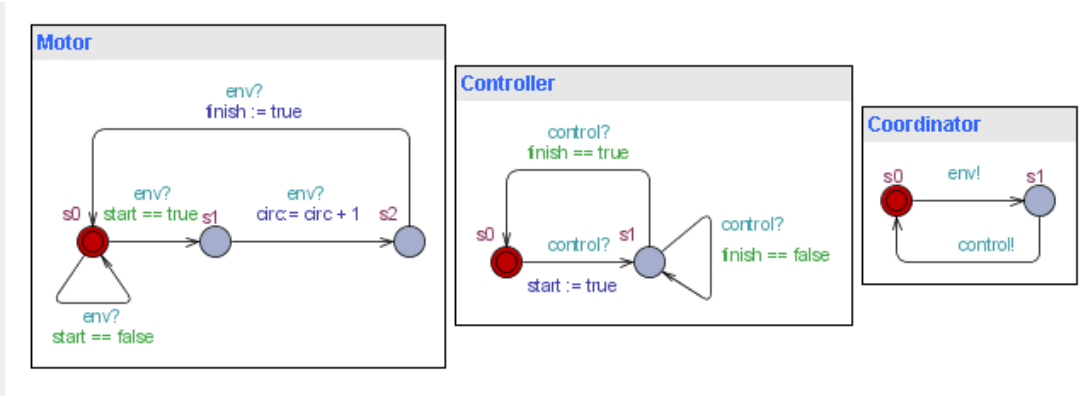

图 2.10　异步执行组件的外部协调同步模式示意图

第二种和第三种模式，则在较低的层次考虑了环境和控制器的交互。第二种模式是在分布式系统编程中经常采用的一种同步方法，但这种模式需要接收方修改信号变量值，不适合PLC应用。例如在电机实例中，电机的*finish*信号是PLC控制器的输入变量， PLC编程时通常不修改其输入变量值，因此这种模式难以与PLC实现相对应。对于第三种模式，从PLC控制器角度看，它在循环周期开始时接收环境输入（此时环境可能执行了一步或者仍然处于当前状态），然后根据环境输入进行响应，或者（当没有期望的输入时）不执行任何动作，接着进入下一个扫描周期，循环往复执行。因此，它与PLC的循环周期运行模式保持一致。我们将在PLC领域的EDOLA设计中考虑第三种环境和PLC交互模式。

2.4.2.4　常见验证需求

根据已有的PLC控制软件应用实例和验证实践，本章总结出五类常见PLC验证需求，包括四类正确性属性和一类用于模型调试的排错属性。按照Pnueli等[104]从语法角度的分类，它们都属于安全属性，分别描述如下。

响应属性　系统应该仅在用户遵循期望的交互顺序在控制面板上输入控制命令时，才做出相应的响应。即对于一个系统动作，若其未处于良好定义的条件下，即使用户输入命令，此动作也不会发生。例如：自动售货机的PLC控制系统，要求在饮料释放过程，面板的所有操作都无效。

竞争属性 这类属性表达多个对象之间的竞争关系，使得同一时刻最多只能有一个对象被选择。例如：控制电机运转的PLC系统，要求不能同时发出正转和反转命令；十字路口交通控制PLC系统，要求任何情况下不会同时亮起行车道和人行道的绿灯。

顺序属性 这类属性表示某一对象的工作要以另一对象的工作为前提。例如：控制液体混合装置的PLC系统，要求混合阀门当且仅当搅拌机工作之后才能打开。

优先属性 在实际系统中，还会有优先级的要求，即某个系统动作比其它系统动作的优先级要高。例如：双模式吊杆的PLC控制系统，要求当用户在控制面板同时按下自动和手动模式按钮时，优先选择手动方式运行。

排错属性 一类常见的排错属性是检查模型中定义的动作、变迁等是否可以发生。因为我们假定模型中定义的每个动作或变迁都有它存在的意义，若它们在系统任何时刻都不能被使能（Enabled），则很有可能模型描述时出现了错误。例如：抢答器PLC控制系统，要求用户抢答、蜂鸣器响、点亮抢答指示灯等动作都可以发生。

2.4.3 领域知识和验证需求表示

与第 2.3 节介绍的生产调度领域知识表示不同，第 2.4.2 节总结的PLC领域知识很难表示为领域操作符的形式。对于这类领域知识，一般通过关键字标识，并实现在EDOLA的语义解释中，在语法中没有直接的操作符对应。譬如PLC领域的输入变量、输出变量和系统变量在EDOLA语法中可分别通过关键字 **INPUTVAR**、**OUTPUTVAR** 和 **SYSTEMVAR** 标识，以便在语义解释时对其进行不同的处理。环境描述部分的全环境可以通过关键字 **ENV TOTAL** 表示，以和自定义环境模型相区分。为便于理解，我们将验证需求的操作符表示与其形式语义一起，在第 2.4.4 节进行介绍。

2.4.4 基于TLA⁺的形式语义

本节介绍第 2.4.2 节的领域知识对应的TLA⁺形式语义，以及验证需求的操作符描述和形式语义。

2.4.4.1 变量类别

EDOLA语法中通过关键字 **INPUTVAR**、**OUTPUTVAR** 和 **SYSTEMVAR**区分的三类变量，分别解释为TLA⁺的三个元组： IV、SV和OV。具体来讲，设通过**INPUTVAR**声明了输入变量$invar_1, \ldots, invar_{n_i}$；通过**OUTPUTVAR**声明了输出变量$outvar_1, \ldots, outvar_{n_o}$；通过**SYSTEMVAR**声明了系统变量$sysvar_1, \ldots, sysvar_{n_s}$，则对应的TLA⁺表示为

$$
\begin{aligned}
IV &\triangleq \langle invar_1, \ldots, invar_{n_i} \rangle \\
OV &\triangleq \langle outvar_1, \ldots, out_{n_o} \rangle \\
SV &\triangleq \langle sysvar_1, \ldots, sysvar_{n_s} \rangle
\end{aligned}
$$

2.4.4.2 环境描述

自定义环境模型被解释为一组TLA⁺动作。它们根据不同的应用，具有不同的模式，不能一概而论。对于**ENV TOTAL**表示的高可信要求下的全环境模型，则可直接对应到一个TLA⁺动作描述模式，表示为

$$
EnvInput \triangleq \bigwedge_{i \in n_i} invar_i' \in ValRange_i
$$

此语义解释使用了变量类别这一领域知识。全环境模型通过描述PLC控制系统的每一个输入变量$invar_i, i = 1, \ldots, n_i$的值都可修改为其取值范围（用$ValRange_i$表示）内的任意值实现。

2.4.4.3 PLC控制系统和环境的交互模式

PLC控制系统和环境的基本交互模式，基于TLA⁺形式定义为

$$
\begin{aligned}
Next \triangleq\ & \lor \land aux = 0 \\
& \quad \land EnvInput \\
& \quad \land \text{UNCHANGED } SOV \\
& \quad \land aux' = 1 \\
& \lor \land aux = 1 \\
& \quad \land SystemAction \\
& \quad \land \text{UNCHANGED } IV \\
& \quad \land aux' = 0
\end{aligned}
$$

其中$SOV \triangleq SV \circ OV$，包括所有的系统变量和输出变量。PLC系统动作（由Next定义）可以是一个环境动作*EnvInput*或者一个系统响应动作 *SystemAction*。*aux*是一个辅助变量，取值为整数，用于实现PLC的循环扫描特征。

当环境采用全模型描述时，由于包括了所有可能的环境输入，因此在验证阶段会带来非常大的状态空间。为了有效验证，在全环境模式下，我们进一步将PLC的循环扫描特征解释为TLA⁺动作

$$
\begin{aligned}
Next \triangleq\ & \lor \land aux = 0 \\
& \quad \land EnvInput \\
& \quad \land \text{UNCHANGED } SOV \\
& \quad \land aux' = 1 \\
& \lor \land aux = 1 \\
& \quad \land SystemAction \\
& \quad \land \text{UNCHANGED } IV \\
& \quad \land aux' = 2 \\
& \lor \land aux = 2 \\
& \quad \land ClearEnvInput \\
& \quad \land \text{UNCHANGED } SOV \\
& \quad \land aux' = 0
\end{aligned}
$$

其中，*ClearEnvInput*动作重置所有输入变量为初始状态值。由于全环境模型中包含了所有可能的输入，而根据PLC的循环扫描周期，每个扫描周期开始时会重新检测环境输入，因此在基本模式上加入*ClearEnvInput*不影响基本模型的语义，而且这个模式与基本模式相比，还会减少验证时的状态空间。

2.4.4.4　验证需求操作符和语义

本章总结的几种PLC领域验证需求都可通过不变式来描述，以 $Inv(P)$ 表示。基于TLA$^+$定义不变式的形式语义时，需要区分P是一个状态谓词或是一个动作。若P表示一个状态谓词，则$Inv(P) \triangleq \Box(P)$；若$P$表示一个动作，则$Inv(P) \triangleq \Box[P]_{vars}$，其中$vars$表示系统所有变量构成的元组。四类正确性验证需求以及一类排错属性都是特殊形式的不变式，它们的操作符描述和形式语义定义如下。

响应属性　动作的有选择响应属性通过操作符 $Respond(Act, EnvState, SysState)$ 表示，它的直观含义是：对于$EnvState$所表示的环境输入，只有当系统满足$SysState$所表示的条件时，动作Act才可能执行。此操作符的形式语义定义为

$$Respond(Act, EnvState, SysState) \triangleq \Box(EnvState \wedge \neg SysState \Rightarrow \neg(\textsc{enabled}\ Act))$$

其中Act是一个动作名，$EnvState$和$SysState$是状态谓词。$\Box$是时序逻辑操作符，$\Box P$表示状态谓词P永远为真。事实上，此公式表达的含义是：若系统不满足$SysState$条件，即使环境输入为$EnvState$，动作Act也不使能，因而不能执行。这是$Respond$操作符的另一种解释。

竞争属性　系统状态之间的竞争属性表达为操作符 $Compete(Cond, State_1, State_2)$，表示在条件$Cond$下，系统状态$State_1$和$State_2$不能同时成立。它的TLA$^+$语义解释为

$$Compete(Cond, State_1, State_2) \triangleq \Box(Cond \Rightarrow \neg(State_1 \wedge State_2))$$

顺序属性　$Sequence(Act, SysState)$ 操作符表达了顺序属性，即动作Act只有在$SysState$条件成立时，才可能执行。$SysState$表示执行某些动作之后得到的系统状态，从而$Sequence$操作符表达了动作Act的前提条件。它被解释为TLA$^+$时序逻辑中的一个蕴含关系

$$Sequence(Act, SysState) \triangleq \Box[Act \Rightarrow SysState]_{vars}$$

其中$vars$表示模型中所有变量组成的元组。

优先属性　通过操作符$Priority(Act, SysState)$表示，即对于用户定义的所有系统动作，在$SysState$ 定义的系统状态下， Act动作具有最高的优先级。它的TLA$^+$解释为

$$Priority(Act, SysState) \triangleq \Box(SysState \Rightarrow \wedge \text{ENABLED } Act \wedge \neg\text{ENABLED } (OtherActions))$$

其中，$OtherActions$表示除 Act 之外的所有其它动作。此公式指定，在 $SysState$ 成立的情况下，只有 Act 动作被使能而所有其它的系统动作都是非使能的（Disabled），因此只有 Act 可能被执行，从而保证了 Act 动作的优先性。

排错属性　通过操作符$Debug(Act)$表示，即检查动作Act是否可以发生。它对应于TLA$^+$的属性定义：

$$Debug(Act) \triangleq \Box[\neg Act]_{vars}$$

与正确性相反，我们期望$Debug$操作符表示的属性验证结果为假，从而表明系统中存在一条路径，使得Act成立，也就是说，系统中存在一条路径，使得Act定义的动作有可能发生。

2.5　本章小结

本章介绍了EDOLA语言领域知识层的研究方法以及在生产调度和PLC控制软件两个典型领域的实践。领域知识层的研究按照领域界定、领域知识提取、语言描述和形式语义定义四个步骤进行。在生产调度算法实践中，提取了生产调

度问题、时间Petri网模型、算法的五个步骤、顺序执行模式等领域特征以及算法的两类验证需求；在PLC控制实践中，提取了变量类别、环境模式、PLC和环境的交互模式等领域特征以及五类常见验证需求。针对提取的领域特征，定义了领域知识描述操作符并基于TLA$^+$给出各操作符（及不能用操作符表达的领域知识）的形式语义。领域知识和验证需求的提取及操作符的定义为EDOLA的语法设计提供了基础；操作符的形式语义为EDOLA语言的语义解释及自动验证提供了支持。

第 3 章　EDOLA的公共模块层研究和实现

EDOLA语言设计的公共模块层，负责对领域“共性”知识的表达。公共模块层所表达的知识，通常可由领域库中的操作完全表示，以供设计具体领域的EDOLA语言时进行选择。本章以实时模块为例，介绍EDOLA公共模块层的设计方法和实践。通过对实时系统实例、已有形式语言和建模语言的分析，本章抽取了时间相关的共性知识，提出了两类基本时间操作符和四类高级时间操作符，并基于TLA$^+$ 给出了各操作符的形式语义。形式语义的定义中考虑了强语义和弱语义两种情况；语义解释方式的选择考虑了易于验证的特征。我们通过一个具体的实例，说明了公共模块层定义的时间操作符在时间特征描述上的易用性。

3.1　引言

EDOLA语言设计的公共模块层位于领域知识层和验证支持层之间，负责表达领域的“共性”知识。公共模块层在传统DSL设计方法学中没有对应的部分。在EDOLA的设计中加入公共模块层，是为了领域EDOLA语言设计的复用性。若仅考虑单个具体领域的EDOLA设计，公共模块层的内容可并入领域知识层统一考虑；但从多个领域的EDOLA语言设计考虑，通过公共模块层的共性知识积累，可避免EDOLA的从零设计，从而降低了EDOLA设计的难度。

公共模块层的设计方法，与领域知识层类似，可基于领域工程或者“程序族”进行领域知识的提取。对EDOLA而言，“程序”指的是基于一种形式语言所描述的系统模型。值得注意的是，公共模块层表示的知识除了要求其“共性”特征之外，我们还要求它们可通过领域库中的操作完全表示。因此，如第 2 章介绍的PLC循环扫描模式以及生产调度算法的顺序执行模式，就不能作为共性特征在公共模块层描述。领域知识层可直接选用公共模块层定义的操作符，或者根据更具体的领域知识，对基本的公共知识操作符进一步封装。此外，由于公共模块层操作符的“共性”，它们的验证支持研究成果可被多个特定领域的EDOLA直接继承。

随着实时系统、嵌入式系统和普适计算的广泛应用，时效性系统（即行为受时间的流逝所影响的系统）受到了越来越多的关注[105,106]。第 2 章所列举的生产调度算法以及许多PLC控制系统都属于时效性系统范畴。为了验证时效性系统，建模语言中需要支持对时间的表达。目前有多种形式语言和建模语言都考虑了时间的扩展，如时间CSP[107]、时间自动机[16]、ET-LOTOS[108]、UML的实时特性描述文件UML-MARTE[54]等。因此，本章以“时间”这一共性特征为例，介绍EDOLA公共模块层的设计方法和实践。

本章的组织结构如下：第 3.2 节介绍时间特征的提取和表示；第 3.3 节介绍时间操作符的形式语义。第 3.4 节给出一个案例分析来解释时间操作符在实际问题中的应用。最后，第 3.5 节是对本章的小结。

3.2 时间特征的知识提取和表示

本章的时间特征知识提取，混成采用了基于领域工程和基于“程序族”的方法。一方面，我们通过分析实践中的各种时间要求和时间约束场景，总结一般性的时间特征；另一方面，参考已有形式语言和建模语言中对时间特征的表达，来完善时间特征的提取。

时间Petri网[109]和时间自动机提供了基本的时间约束表达方法。时间Petri网通常在变迁上添加时间信息，又分为表达变迁执行时间的确定值以及时间范围两种；时间自动机在状态上表达时间的演进，既可表示对单个变迁的执行时间约束，也可表示对多个变迁执行总体时间的约束。时间CSP和TCOZ[110]等语言提供了高级的时间描述原语，如时延（Delay）、超时（Timeout）、最后期限（Deadline）等，更适合于领域问题的描述。 Konrad等[111]提出了嵌入式系统建模和分析中常见的时间模式及其结构化自然语言表示。董劲松等[112]给出了时间CSP和TCOZ语言中的高级时间原语所对应的时间自动机模式，以便提供这些时间原语的验证工具支持。UML-MARTE是从时间特征和UML角度提出的关于表达时间相关概念的一般框架。它将时间模型划分为时间演进的描述、时间值或时间区间的访问以及时间的使用三类。

参考已有研究结果，本章总结了实时模型中的常见时间特征及其关系，基于特征图表示为图 3.1 。

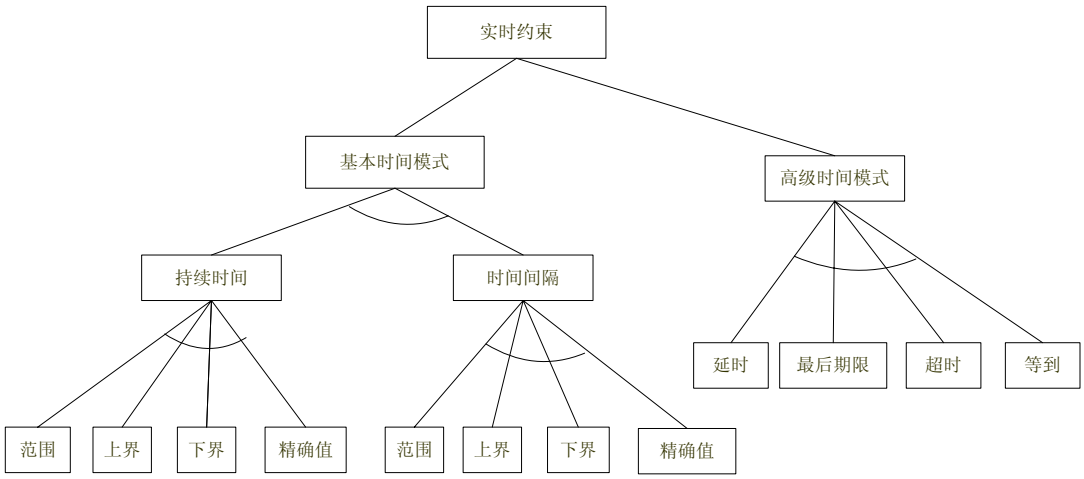

图 3.1　时间特征图

时间特征分为基本时间模式和高级时间模式两类，而基本时间模式又分为单个动作的持续时间以及动作间的时间间隔两种情形。下面分别对动作持续时间模式、动作之间时间间隔模式以及高级时间模式进行介绍，并给出它们的公共操作符表示。

3.2.1　动作持续时间

对动作执行持续时间的约束在时效性系统中很常见。例如，要求一个只有小时刻度的电子时钟移动一格的时间应在59.9分钟到60.1分钟之间。时间约束又可分为时间范围约束、时间上限约束、时间下限约束和时间值约束四种。因此，本节提供四个操作符，以分别描述单个动作持续时间的四种可能约束。这四个操作符的名字和基本参数为

$$DurationBound(STRflag, t, A, v, min, lb, max, ub)$$

$$DurationUB(STRflag, t, A, v, max, ub)$$

$$DurationLB(t, A, v, min, lb)$$

$$DurationValue(STRflag, t, A, v, val)$$

直观来看，$DurationBound$表示要求动作A执行的持续时间满足范围区间$[min, max]$。参数lb和ub是用于区分此区间为开区间或闭区间的布尔标识。当lb取值为真时，min被包含在范围区间中；而当ub取值为真时，max被包含在允许的范围区间中。其它参数如$STRFlag$、t和v与操作符的形式语义相关，将在第 3.3 节介绍。$DurationUB$表示当只有动作A的持续时间上限被约束的情形；而$DurationLB$表示当只有时间下限被约束的情形；$DurationValue$则表示要求动作A的持续时间为一个精确值val。

3.2.2 动作间时间间隔

在时效性系统中，对具有先后关系的两个动作的时间间隔约束同样很常见。譬如，要求动作B和动作D之间的时间间隔应总是大于2.5秒；要求动作C在动作A发生后3秒之内发生。为标准化这类时间约束，本节也提供了四个操作符，分别描述对动作间时间间隔的不同要求。它们的名字和基本参数表示为

$$IntervalBound(STRflag, t, Acts, B, v, min, lb, max, ub)$$

$$IntervalUB(STRflag, Acts, B, v, max, ub)$$

$$IntervalLB(Acts, B, v, min, lb)$$

$$IntervalValue(STRflag, Acts, B, v, val)$$

为表示两个动作A和B之间的时间间隔，我们通过考虑A和B之间的动作来定义。参数$Acts$用于记录动作A和B之间的所有动作。$IntervalBound$表示动作时间间隔的范围约束，参数定义与动作持续时间的对应参数相同。$IntervalUB$表示只考虑时间间隔上限约束的情形；$IntervalLB$则表示只有时间间隔下界被要求的情形；$IntervalValue$表示对时间间隔的精确时间要求。

3.2.3 高级时间模式

基本时间模式提供了时间约束在公共模块层的基本表达方法。在特定领域EDOLA设计时，可根据领域要求，将这些基本操作符封装为更高层的领域知识操作符。在实践中，研究者们已总结出一些常用的时间模式，如时延、超时、最后期限等，因此我们在公共模块层也提供了一些高级时间操作符，以便

为EDOLA设计提供更多的选择。目前已定义的四个高级时间操作符及其参数为

$$Delay(A, time, t, v)$$
$$Deadline(A, time, t, v)$$
$$Timeout(A, B, time, t1, t2, v)$$
$$WaitUntil(A, Idle, time, t, v)$$

其中，$Delay$是*时延*操作符，表示动作A在执行时间$time$之后结束；$Deadline$ 是*最后期限*操作符，表示动作A必须在时间$time$之前完成； $Timeout$是*超时*操作符，它的含义是：若动作A未在时间$time$内执行，则执行动作B； $WaitUntil$是等*到*（Wait-Until）操作符，它的精确语义是：若动作A的执行时间小于$time$，则保持空闲等待，直到时间达到数值$time$之后才继续后续动作。$Idle$ 表示动作A和它的后续动作之间的空闲动作，参数t，$t1$，$t2$和v与形式语义定义相关，将在第 3.3 节介绍。

3.3 操作符的形式语义

本节基于TLA$^+$，给出各个基本时间操作符和高级时间操作符的形式语义。为了在TLA$^+$ 中表示这些时间概念，我们定义了一个TLA$^+$的实时模块*RealTimeNew*，所有时间操作符的形式语义都被解释为*RealTimeNew*中的动作定义。

在时间建模方面，Ababi等[113]首先提出了在TLA逻辑中建模时间的一般形式。与许多隐式时间模型如时间自动机和时间Petri网不同，他们显式使用一个状态变量*now*来表示时间，并通过表示“动作在正确的时刻发生”来描述期望的系统行为。之后，此时间描述形式被研究者们所采用，以描述各种时效性系统[114–116]。为标准化TLA$^+$的实时描述，Lamport[26] 开发了一个实时模块*RealTime*，给出了单个动作持续时间的约束描述模板，但动作间时间间隔的表达并未被考虑进来。

本章的*RealTimeNew*模块的设计受到了*RealTime*模块的启发，但这两个模块又有着显著的区别：在*RealTime*模块中，时间的演进是一个完整的规范说明，而动作上的时间约束同样表达为一个完整的规范说明，因此需要通过规范说明组合 [26]来使用；而我们设计的*RealTimeNew* 模块，由于以解释时间操作符为目

的，因此各种时间约束以及时间的演进都通过动作定义来表达。我们的这种设计也考虑了自动验证的因素。通过验证支持层的实现，采用这种TLA⁺语义解释的EDOLA模型可直接被TLA⁺的模型检测工具TLC所处理或进行演绎推理，而基于*RealTime*模块定义的格式则不可以。

为了解释各时间操作符，需要在TLA⁺中描述时间的演进方式。本节首先介绍时间演进在 TLA⁺中的表示，再分别介绍动作持续时间操作符、时间间隔操作符以及高级时间操作符的形式语义。

3.3.1　时间演进

时间演变的描述与*RealTime*模块中的对应部分基本相同。实际时间通过一个实数值变量*now*表达。*RealTimeNew*模块中使用两个动作来描述时间的演变。时间变量*now*的改变，定义为动作

$$NowNext(v) \triangleq \land\ now' \in \{r \in Real : r > now\}$$
$$\land\ \text{UNCHANGED}\ v$$

其中，now'表示变量*now*在执行一个动作后，所得新状态中的值；*Real*代表所有实数的集合。动作*NowNext(v)*将now的值改变为一个满足 $r > now$的任意实数值r，并保持参数v的值不变。在实际应用时，参数v将使用变量名或者变量元组名来调用，表示时间演变动作执行时，v所代表的所有变量值保持不变。

此外，时间*now*的值应可以无限增长（这是实际的时间演变性质），因此我们需要给出一个公平性约束，来排除时间的“Zeno”行为[113]。在*RealTimeNew*模块中，此公平性约束由下面的公式定义

$$RTFairness(v) \triangleq \forall r \in Real : WF\_now(NowNext(v) \land (now' > r))$$

其中，$WF_{var}(A)$形式的公式表示动作A上的弱公平性约束（Weak Fairness），它在TLA⁺中的定义为：$\Box(\Box\text{ENABLED}\ \langle A\rangle_{var} \Rightarrow \Diamond\langle A\rangle_{var})$，含义为：若动作$A$被连续使能（Continously Enabled），则它最终会发生。由于动作$NowNext(v) \land (now' > r)$总是使能的，因此$RTFairness(v)$事实上表达了：任给一个实数$r$，*NowNext(v)*动作最终将执行，且变量*now*的新值会大于r，从而保证了时间行为的非Zeno性质。

3.3.2 动作持续时间的语义

公共知识表示部分提供了动作持续时间的四个基本操作符。我们首先考虑范围操作符：*DurationBound(STRflag, t, A, v, min, lb, max, ub)*，它表示对动作A持续时间的约束是一个由上限和下限构成的范围约束。为给出此操作符的精确语义，必须考虑的一个问题是：随着时间的演进，当到达时间约束上限时，是否要求动作A必须发生？根据一般的约定，若要求到达时间上限时动作A必须发生，则采用的是*强语义*解释; 否则，采用的是*弱语义*解释[117]。在实际应用中，这两种时间语义的要求都存在，因此在解释$DurationBound$操作符时，我们将这两种语义都考虑进来，并通过布尔标识$STRflag$进行区分。$DurationBound$操作符的TLA⁺语义定义为

$$
\begin{array}{l}
DurationBound(STRflag, t, A, v, min, lb, max, ub) \triangleq \\
\quad \textsc{let}\ TNext \triangleq t' = \textsc{if}\ \langle A\rangle_v \vee \neg(\textsc{enabled}\ \langle A\rangle_v)' \\
\qquad\qquad\qquad\qquad \textsc{then}\ 0 \\
\qquad\qquad\qquad\qquad \textsc{else}\ t + (now' - now) \\
\qquad UpperBound \triangleq \textsc{if}\ STRflag = \textsc{true} \\
\qquad\qquad\qquad\qquad \textsc{then if}\ ub\ \textsc{then}\ t' \leq max\ \textsc{else}\ t' < max \\
\qquad\qquad\qquad\qquad \textsc{else}\ A \Rightarrow \textsc{if}\ ub\ \textsc{then}\ t \leq max\ \textsc{else}\ t < max \\
\qquad LowerBound \triangleq A \Rightarrow \textsc{if}\ lb\ \textsc{then}\ t \geq min\ \textsc{else}\ t > min \\
\quad \textsc{in}\ \ \wedge TNext \\
\qquad \wedge UpperBound \\
\qquad \wedge LowerBound
\end{array}
$$

$DurationBound$被解释为一个具有八个参数的TLA⁺动作。第一个参数是强语义标识$STRflag$，它的取值为TRUE时表示强语义解释，而取值为FALSE时表示弱语义解释。A表示要关注的动作；t是记录动作A持续时间的时钟，它应满足的范围区间为$[min, max]$。TLA⁺要求显式给出时钟的名字。参数v表示除时间变量*now*和时钟变量之外的所有变量（即功能变量），它是采用TLA⁺作为基语言进行语义解释时需提供的必要参数。

整个公式由三个子公式*TNext*、*UpperBound*和 *LowerBound*合取而成。LET IN 结构用来将三个子公式的定义局部化在*DurationBound*的定义中。*TNext*指定

时钟t的值如何改变。根据定义，当动作A发生或下一步由于执行某动作而使得动作A变为非使能时，时钟 t的新值被置为零。否则，表示动作A被连续使能，时钟t 的新值将随着时间的演变而逐渐累加。$\langle A\rangle_v$等价于$A \wedge v' \neq v$, 表示动作A的实际发生，排除了系统的停滞（Stuttering）步骤。*LowerBound*指定，若动作A发生，则时钟t的当前值必须大于（或等于）min，从而表达了范围的下限要求。*UpperBound*指定了时间范围的上限要求，区分强语义和弱语义两种情形。在强语义约束下，公式指定时钟t的新值必须无条件小于（或等于）max，从而保证了动作A在其时钟t的值到达上限时，必须发生。在弱语义约束下，公式仅指定，若动作A发生，则时钟t的当前值必须小于（或等于）max。因此，当上限到达时，动作A并不要求发生，只是（按照规定）当时间超过上限时，动作A就不再可能发生了。综上所述，合取这三个子公式所得公式*DurationBound*，指定了动作A持续时间的一个范围约束。

根据$DurationBound$的TLA$^+$语义解释，第 3.2.1 节的电子时钟例子，可用*DurationBound*操作符表示为：*DurationBound(*TRUE*, t, Step, v, 59.9,* FALSE*, 60.1,* FALSE*)*，其中 *t*是时钟变量, *v*是变量元组，*Step* 表示时钟移动一格这个动作。

其它三个操作符的语义定义分别是*DurationBound*语义定义的三个变体。操作符*DurationUB*表达了当只有持续时间上界被约束的情形，其语义同样被解释为一个TLA$^+$动作，并区分为强语义和弱语义两种情形，定义为

$$
\begin{array}{l}
DurationUB(STRflag, t, A, v, max, ub) \triangleq \\
\quad \text{LET } TNext \triangleq t' = \text{IF } \langle A\rangle_v \vee \neg(\text{ENABLED } \langle A\rangle_v)' \\
\qquad\qquad\qquad\qquad \text{THEN } 0 \\
\qquad\qquad\qquad\qquad \text{ELSE } t + (now' - now) \\
\qquad UpperBound \triangleq \text{IF } STRflag = \text{TRUE} \\
\qquad\qquad\qquad\qquad \text{THEN IF } ub \text{ THEN } t' \leq max \text{ ELSE } t' < max \\
\qquad\qquad\qquad\qquad \text{ELSE } A \Rightarrow \text{IF } ub \text{ THEN } t \leq max \text{ ELSE } t < max \\
\quad \text{IN } \wedge TNext \\
\qquad\quad \wedge UpperBound
\end{array}
$$

可以看出，与操作符*DurationBound*相比，*DurationUB*的定义只是略去了对下限的约束，其它部分完全相同。与之对应，我们可定义操作符*DurationLB(t, A,*

*v, min, lb)*的语义解释。由于只表示下限要求，因此不涉及强语义和弱语义的区分。它对应于 *DurationBound*定义中的*TNext*和*LowerBound*部分，在此不再赘述。

操作符*DurationValue*用于表达对动作持续时间的精确时间要求。它的语义可基于 *DurationBound*操作符给出

$$DurationValue(STRflag, t, A, v, val) \triangleq$$
$$DurationBound(STRflag, t, A, v, val, \text{TRUE}, val, \text{TRUE})$$

其中*val*是要求动作A和B的时间间隔应满足的具体值。

3.3.3 动作间时间间隔的语义

本节定义四个时间间隔操作符的形式语义。第一个操作符是 *IntervalBound(STRflag, t, Acts, B, v, min, lb, max, ub)*，它指定时间间隔所允许的范围。与定义单个动作持续时间的形式语义类似，$IntervalBound$操作符的语义同样需区分强语义和弱语义两种情况。具体来讲，当考虑两个具有先后关系的动作A和B之间的时间间隔时，需要明确：随着时间的演进，当范围的上限达到时，是否要求动作B必须发生？当采用强语义时，答案是肯定的；而采用弱语义时，答案是否定的。对于第 3.2.2 节举的两个例子，前者表达了一个弱语义约束，后者表达了一个强语义约束。不同的应用，会在强语义和弱语义之间有不同的选择。因此，时间间隔操作符的形式语义定义时同样考虑了两种语义解释。

由于单个动作持续时间的存在（见第 3.2.1 节所述），进行形式描述时，另一个必须考虑的问题是：当定义动作A和B的时间间隔约束时，是考虑A和B的开始执行还是执行完毕？这里也存在多种选择的可能。本章选用的时间间隔语义是：从动作A执行完毕开始，到动作B执行完毕为止这个时间段上的约束。综上分析，我们可将操作符$IntervalBound$ 的形式语义定义为

$$IntervalBound(STRflag, t, Acts, B, v, min, lb, max, ub) \triangleq$$
$$\begin{aligned}&\text{LET } TNext \triangleq t' = \text{IF } \langle B\rangle_v \vee \neg(\text{ENABLED } \langle Acts \vee B\rangle_v)'\\ &\qquad\qquad \text{THEN } 0\\ &\qquad\qquad \text{ELSE } t + (now' - now)\end{aligned}$$

$$
\begin{aligned}
&\mathit{UpperBound} \triangleq \text{IF } \mathit{STRflag} = \text{TRUE} \\
&\qquad\qquad \text{THEN IF } \mathit{ub} \text{ THEN } t' \leq \mathit{max} \text{ ELSE } t' < \mathit{max} \\
&\qquad\qquad \text{ELSE } B \Rightarrow \text{IF } \mathit{ub} \text{ THEN } t \leq \mathit{max} \text{ ELSE } t < \mathit{max} \\
&\mathit{LowerBound} \triangleq B \Rightarrow \text{IF } \mathit{lb} \text{ THEN } t \geq \mathit{min} \text{ ELSE } t > \mathit{min} \\
\text{IN} \quad &\wedge \mathit{TNext} \\
&\wedge \mathit{UpperBound} \\
&\wedge \mathit{LowerBound}
\end{aligned}
$$

计算动作A和B之间时间间隔的主要思路是，通过指定从动作A执行完毕到动作B执行完毕的时间总和来表示。计算这个时间，需要得知哪些动作发生在A和B之间。由于我们考虑的是具有先后关系的两个动作，从而使得获得这一信息成为可能。设有n个动作发生在动作A和B之间，按顺序表示为：$A_1, \ldots, A_n$，则在使用操作符$\mathit{IntervalBound}$表达A和B之间的时间间隔约束时，取$\mathit{Acts} \triangleq A_1 \vee \ldots \vee A_n$即可。此时间间隔通过时钟$t$来记录。其它参数如$\mathit{STRflag}, \mathit{max}, \mathit{min}$等与它们在$\mathit{DurationBound}$定义中的含义相同。

整个公式在形式上与$\mathit{DurationBound}$很相似，但在时钟t的计时上即TNext的表达上非常不同。根据TNext的定义，当动作B发生或者下一步由于执行某动作而使得动作$\mathit{Acts} \vee B$变为非使能时，时钟 t的新值被置为零。否则，时钟t 的值随时间演变而累加。它表达的直观含义是，只对与此时间间隔相关的部分进行计时，而对所有其它动作（既不使能A和B之间的某动作，又不使能动作B），时钟t不计时。其它解释与$\mathit{DurationBound}$类似。

例如，已知动作A、B、C和D顺序发生。若要求动作B和D之间的时间间隔大于或等于2.5秒且小于3.5秒，则此时间约束可由实例化的公式 *IntervalBound(*TRUE*,t, C, D, v, 2.5,* TRUE*, 3.5,* FALSE*)*表示，其中 t 是记录此时间间隔的时钟，v 表示和时间演变无关的变量元组。

值得说明的是，当要考虑的两个动作A和B相邻发生时，根据我们采用的时间间隔语义，它事实上表示了动作B的持续时间。在形式语义定义中，由于A和B相邻发生，因此 Acts = FALSE，从而，操作符$\mathit{IntervalBound}$的语义定义，退化为了与$\mathit{DurationBound}$操作符完全相同的形式。这一点同上述对实际情况的分析一致。

其它三个操作符的语义定义分别是*IntervalBound*操作符语义定义的三个变体，具体描述略。

3.3.4　高级时间模式的语义

第 3.2.3 节给出了四个高级时间操作符，它们可完全通过基本时间操作符进行解释。本节介绍它们基于基本操作符的语义定义。

时延　*Delay*操作符表示动作A在执行时间$time$之后结束，因此可使用基本的持续时间值操作符*DurationValue*，定义为

$$Delay(A, time, t, v) \triangleq DurationValue(\text{TRUE}, t, A, v, time)$$

其中t表示记录此延时的时钟，v表示所有的功能变量，这两个参数是TLA$^+$表达的要求。

最后期限　*Deadline*操作符表示动作A必须在时间$time$之前完成，因此可采用基本的持续时间上限操作符*DurationUB*表示，定义为

$$Deadline(A, time, t, v) \triangleq DurationUB(\text{TRUE}, t, A, v, time, \text{TRUE})$$

超时　*Timeout*操作符表示若一个动作A未在时间$time$之内执行，则执行另一动作B。它可使用两个基本时间操作符定义为

$$\begin{aligned} &Timeout(A, B, time, t1, t2, v) \triangleq \\ &\wedge\ DurationUB(\text{FALSE}, t1, A, v, time, \text{TRUE}) \\ &\wedge\ DurationValue(\text{TRUE}, t2, B, v, time) \end{aligned}$$

也就是说，*超时*概念被解释为动作A和B上的两个约束。首先，要求若动作A发生，则对应时钟$t1$必须满足$t1 \leq time$，但并不要求动作A发生，这一层含义可通过$DurationUB$操作符的*弱语义*解释表示。另一方面，当时间达到$time$时，动作B必须发生，这段时间通过时钟$t2$记录，可通过基于*强语义*应用$DurationValue$操作符得到。

等到　*WaitUntil*操作符表示若动作A的执行时间小于$time$，则保持空闲等待，直到时间$time$达到后才继续后续动作。它可以通过基本时间操作符*IntervalLB*进行定义

$$WaitUntil(A, Idle, time, t, v) \triangleq IntervalLB(t, A, Idle, v, time, \text{TRUE})$$

3.4　案例分析

本节通过一个抢答实例，介绍如何使用基本和高级时间操作符，完成实际问题的描述。在一个答题比赛中，有一个主持人和多个答题者。比赛规则定义如下：

1. 主持人和答题者分别通过对应的按钮来操作；
2. 主持人按下按钮之后，新一轮的比赛开始；
3. 在一轮比赛开始之后，如果某个答题者在5秒之内按下按钮答题，则对应的指示灯亮，进入答题阶段。否则，蜂鸣器持续响3秒以表示此轮抢答结束；
4. 在答题阶段，选手要回答固定数目的题目。每答对一题，加20分，答错不扣分。整个答题阶段不得超过60秒。答题结束后，主持人可按下按钮开始新一轮的比赛。
5. 在一轮比赛中，只有最先按下按钮的抢答有效。
6. 为了保证公平性，如果有多个抢答者同时按下按钮，则所有抢答都无效。

从抢答实例的自然语言描述可以看出，此系统的时间约束有三个：抢答的5秒超时约束、蜂鸣器响3秒的精确约束以及答题阶段的60秒上限约束。经过分析，这三个时间约束可方便得基于本章所定义的基本和高级时间操作符进行表达：

1. $\forall i \in Competitors : Timeout(Compete(i), BeeperRing, 5, t1, t2, vars)$
2. $Delay(BeeperRingFinish, 3, t3, vars)$
3. $\forall i \in Competitors :$

$$IntervalUB(\text{TRUE}, t4, Answer(i), AnswerFinish(i), 60, \text{TRUE}, vars)$$

第一个时间约束表达关于抢答动作的一个超时约束，它使用了高级时间操作符*Timeout*。*Compete(i)*表示答题者i进行抢答的动作；*Competitors*是一个集合，表

示所有抢答者；*BeeperRing*是蜂鸣器开始鸣响的动作，超时的时间要求是5秒，涉及到的两个动作分别使用时钟$t1$和$t2$计时。第二个时间约束表达了蜂鸣器的一个延时，它使用了高级时间操作符*Delay*进行表达，其中$BeeperRingFinish$ 表示蜂鸣器结束鸣响的动作，使用时钟$t3$计时。第三个时间约束，表达了答题过程不能超过 60秒，它采用了基本时间操作符*IntervalUB*进行表达。*Answer(i)*表示答题者i开始答题的动作，而*AnswerFinish(i)*表示答题结束的动作；此时间间隔通过时钟$t4$进行记录。

3.5　本章小结

本章以时间特征为例，介绍了EDOLA语言公共模块层的设计和实践。通过时间特征的知识提取，定义了两类基本时间操作符和四类高级操作符，便于描述各种常见的时间约束，包括动作的持续时间约束、动作间时间间隔约束以及延时、最后期限、超时及等到等更高层的时间概念。基于TLA$^+$ 给出了各时间操作符的形式语义。操作符的语义定义区分强语义和弱语义两种情形，语义表达形式上考虑了易于验证的特征。通过一个具体的实例，说明了各种时间操作符在时间特征描述上的易用性。时间特征的语义解释被封装在一个TLA$^+$的实时模块中，每个时间操作符被解释为此模块中的一个动作定义。因此，此公共模块层研究框架易于扩展新的时间操作符。其它的公共知识亦可通过定义模块和动作公式的方式进行组织。

第 4 章　EDOLA的验证支持层研究和实现

本章介绍EDOLA验证支持层的研究和实现。验证支持层通过EDOLA到模型检测和自动定理证明工具输入语言的转换，实现自动验证。本章提出基于中间语言的转换方法，实现了多个EDOLA的验证支持重用。中间语言选用形式语言TLA$^+$的子集subTLA，保证了自动验证性。EDOLA到subTLA的编译过程中，提出了面向验证需求的抽象策略，以减少验证过程中的状态空间和搜索空间。在subTLA的自动定理证明方法中，制定了subTLA的关键语法成分：集合、函数等表达式到一阶逻辑的有效编码规则，使得转换后的subTLA模型能够被SPASS等自动定理证明工具处理，从而实现对无限空间模型的自动验证。

4.1　引言

自动验证技术是形式验证实用化的重要手段。目前的自动验证方法主要有模型检测和自动定理证明两种。 EDOLA的验证支持层负责实现语言的自动验证。本章定义EDOLA模型到模型检测以及自动定理证明工具输入语言的转换，从而基于已有的自动验证工具对EDOLA模型进行验证。

EDOLA的验证支持层与传统DSL设计方法学中的语言实现阶段对应。DSL实现阶段的任务是构造DSL的编译器或解释器，通过预处理、嵌入以及扩展已有编译器 / 解释器等实现模式，实现DSL的执行。如何设计一些优化策略，使得编译后的代码可以高效执行，是DSL实现阶段的一个重点。从这个角度考虑，EDOLA的验证支持层相当于提供一个EDOLA语言到自动验证工具的编译器，只是目标不是“执行”而是“验证”。因此，如何设计EDOLA的编译器，以确保得到的自动验证工具输入模型能够被已有工具有效验证，是验证支持层研究的一个重点。

EDOLA通过封装领域知识为语言上的操作符，将领域语义表达在语言的语法结构中，从而实现领域知识描述的易用性。面向不同的应用领域，EDOLA的语法元素可以多种多样。若对每一领域的 EDOLA都要实现一套语言到验证工具的转换方法，则实现代价太大，会减弱EDOLA的优势。因此，本章提出基于中间

语言的验证支持方法，以提高EDOLA验证支持的重用性。验证支持层的实现分为上下两层：上层实现EDOLA到中间语言的转换，下层实现中间语言到自动验证工具的转换。良好定义的下层实现，可以使特定领域的EDOLA语言验证支持层，只在上层实现有所不同，其下层实现可在多个EDOLA语言之间重用。

中间语言一般使用通用形式语言，作为领域专用语言和验证工具专用描述语言的桥梁。EDOLA的领域知识层和公共模块层设计中，均使用了TLA$^+$来描述各操作符的形式语义，因此，验证支持层选用TLA$^+$作为中间语言，直观且易实现。然而，TLA$^+$语言虽然表达能力强，但语法结构复杂，完整的TLA$^+$语言不可能完全自动验证，因此，我们选取TLA$^+$的一个可自动验证子集subTLA，作为验证支持层的中间语言。

本章的研究内容组织如下：第 4.2 节介绍中间语言subTLA；第 4.3 节介绍从EDOLA到subTLA的编译方法；第 4.4 节介绍从subTLA到自动验证工具的转换方法；第 4.5 节是对本章的小结。

4.2　subTLA介绍

subTLA是TLA$^+$语言的可自动验证子集，它的理论基础同样是TLA逻辑、一阶逻辑和集合论。subTLA语法结构的定义，综合了模型检测和自动定理证明的要求。非形式化而言，subTLA 继承了TLA$^+$的模块扩展（EXTENDS）、常量定义（CONSTANT）、变量定义（VARIABLE）、状态函数以及状态谓词定义和动作定义；舍去了原TLA$^+$的模块实例化（INSTANCE）、局部定义（LOCAL）、变量隐藏（∃）、自定义前缀、中缀和后缀操作符定义等语法成分。另外，TLA$^+$ 有非常丰富的表达式定义，而subTLA语言，从自动推理角度出发，选取了部分表达式或者表达式的特殊形式作为subTLA的表达式。表达式的定义见第 4.4 节。

subTLA的一个规范说明由下面的范式构成：$Spec \triangleq Init \wedge \Box[Next]_{vars} \wedge L$，其中 $Init$是一个状态谓词，描述系统所有可能的初始状态。$[Next]_{vars}$是下一个动作定义，它指定系统的每一步（即一个系统运行的每一对相邻状态）满足动作定义$Next$或保持$vars$（系统中的所有变量）不变。允许停滞是TLA逻辑的一个特色，也是保持规范说明组合性的关键。然而，这意味着规范说明将允许系统永远停滞在某个状态，而这通常不是我们所期望的，因此，可通过第三部分的时序逻

辑公式L，指定规范说明的公平性约束，以排除无限停滞的情形。

4.3　EDOLA到subTLA的编译

EDOLA领域知识层和公共模块层操作符的语义解释是基于TLA$^+$给出的，而语义解释所用到的TLA$^+$语法元素都被包含在subTLA之中，因此实现从EDOLA到subTLA的直观编译是容易的，但如何设计编译过程的一些优化策略，使得编译后的subTLA模型能够被自动验证工具有效验证，仍是EDOLA验证支持层研究的重要问题。基于模型检测和自动定理证明技术进行验证时，分别面临状态空间和搜索空间过大的困难，而抽象（Abstraction）通常被认为是解决此问题的一种重要技术[118–120]。因此，本章将抽象策略作为EDOLA到subTLA编译过程中的一个主要优化策略。

EDOLA的语言设计将领域知识层和公共模块层区分开来。本章提出的面向验证需求的抽象策略，会根据不同的验证需求，生成不同的抽象模型。具体来讲，EDOLA到subTLA的编译过程，对领域知识层和公共模块层有不同的处理：若验证需求描述不涉及公共模块层的操作符，则编译为领域知识层对应的subTLA抽象模型；否则，编译为包含领域知识层和公共模块层内容的subTLA完整模型。本章将通过证明完整模型和抽象模型之间的精化（Refinement）关系，保证抽象的正确性。

我们在第 3 章，以实时特征为例，介绍了公共模块层的设计和实现方法。下面基于实时公共模块层的实现，介绍融入了面向验证需求抽象策略的EDOLA到subTLA编译框架以及抽象策略的正确性证明。

4.3.1　EDOLA到subTLA的编译框架

编译框架的基本思路是：首先将领域知识层操作符编译为subTLA的一个领域描述模块$Spec$；然后，若验证需求包含时间特征，则在$Spec$基础上，扩展对实时公共模块层操作符的编译，得到最终的subTLA模型$RTSpec$，否则，直接取Spec作为最终的subTLA模型。假定系统的功能描述部分$Spec$被描述在一个subTLA模块*DomainModule*中，并具有以下形式：$Spec \triangleq Init \wedge \Box[Next]_{vars} \wedge L$，其中$vars$是系统中所有领域相关的状态变量构成的元组。

当验证需求包含时间特征时，包含n个时间约束的subTLA模型，可通过如图 4.1 所示的框架组合而成。此框架基于*DomainModule*模块中的定义以及用于解释实时特征语义的*RealTimeNew*模块中的定义构成。在完整的subTLA模型*RTmodule*中，“EXTENDS M” 语句将模块M中的所有声明和定义都添加到包含这条语句的模块中，此特征与C 语言预处理器的“#$include$”指令类似。除领域描述模块*DomainModule*和时间模块*RealTimeNew*外，TLA$^+$中的预定义模块*Sequences*被包含进来，用来对元组进行操作。EDOLA中的n个时间约束在subTLA中由n个变量记录，它们通过VARIABLE 语句进行声明。完整模型的定义*RTspec*由初始谓词*BigInit*，动作定义*BigNext*和公平性约束*RTL*所构成。

完整模型的初始状态由领域模块的初始状态*Init*、时间变量*now*的初始状态以及n个时钟的初始状态组合而成。组合操作在subTLA中由公式合取表示，见初始谓词*BigInit*的定义。

完整模型的动作定义*BigNext*由领域模块的动作定义*Next*、时间的动作定义*NowNext*以及所有时间约束组合而成。通过将功能部分描述为 $Next$ 和 UNCHANGED $vars$ 的析取式；将时间演进描述为 $NowNext(vars)$ 与 UNCHANGED now 的析取式，模型中定义了功能描述和时间演进的组合方式。回顾$NowNext$的定义，它要求当时间变量Now改变时，所有功能变量$vars$ 保持不变，因此我们这里采用的是交替（Interleaving）的时间语义[121]。时间约束通过调用实时公共模块操作符来描述。此外，如果功能描述是参数化的，时间约束还可通过有界的全称量词形式进行限定：“$\forall v_1 \in S_1, \ldots, v_k \in S_k :$”，其中$S_1, \ldots, S_k$ 是表示约束变量取值范围的集合。在图 4.1 描述的实时组合框架中，借用了正则表达式中的标准符号“?” 表示可选（Optional），“|”符号表示选择（Choice），“+”符号表示一次或多次重复。A_p 和 A_q 表示与时间约束相关的动作（*DomainModule*模块中定义）；t_i 和 t_j 表示时钟变量，$i \in 1..n$ 且 $j \in 1..n$。

元组*RTvars*表示完整subTLA模型的所有变量集，它包括时间变量*now*，n个时钟以及领域描述模块中的所有变量。$RTvars$ 的定义使用元组组合操作符 $\circ$ 表示。

最后，公平性约束RTL由领域描述部分的公平性约束L和时间演变的公平性约束*RTFairness(vars)*组合而成。

MODULE *RTmodule*

EXTENDS $FuncModule, RealTimeNew, Sequences$

VARIABLE $t_1, t_2, \ldots, t_n$

$$
\begin{aligned}
BigInit \triangleq\ & \wedge Init \\
& \wedge now = 0 \\
& \bigwedge_{i=1}^{n} t_i = 0
\end{aligned}
$$

$$
\begin{aligned}
BigNext \triangleq\ & \wedge (Next \vee (\text{UNCHANGED } vars)) \\
& \wedge (NowNext(vars) \vee \text{UNCHANGED } now) \\
& (\wedge (\forall v_1 \in S_1, \ldots, v_k \in S_k)? \\
& \qquad DurationBound(STRflag, t_i, A_p, vars, min, lb, max, ub_i) \\
& \quad | \quad DurationUB(STRFlag, t_i, A_p, vars, max, ub) \\
& \quad | \quad DurationLB(t_i, A_p, vars, min, lb) \\
& \quad | \quad DurationValue(STRflag, t_i, A_p, vars, val) \\
& \quad | \quad IntevalBound(STRflag, t_i, A_q, A_p, vars, min, lb, max, ub) \\
& \quad | \quad IntervalUB(STRflag, t_i, A_q, A_p, vars, max, ub) \\
& \quad | \quad IntervalLB(t_i, A_q, A_p, vars, min, lb) \\
& \quad | \quad IntervalValue(STRflag, t_i, A_q, A_p, vars, val) \\
& \quad | \quad Delay(A_p, time, t_i, vars) \\
& \quad | \quad Deadline(A_p, time, t_i, vars) \\
& \quad | \quad Timeout(A_p, A_q, time, t_i, t_j, vars) \\
& \quad | \quad WaitUntil(A_p, Idle, time, t_i, vars))^+
\end{aligned}
$$

$$RTvars \triangleq \langle now, t_1, \ldots, t_n \rangle \circ vars$$

$$
\begin{aligned}
RTL \triangleq\ & \wedge L \\
& \wedge RTFairness(vars)
\end{aligned}
$$

$$RTspec \triangleq BigInit \wedge \Box[BigNext]_{RTvars} \wedge RTL$$

图 4.1　基于subTLA的实时组合框架

4.3.2　抽象策略的正确性证明

本节将证明，基于第 4.3.1 节的编译框架得到的完整subTLA模型是领域知识层描述对应的抽象subTLA模型的一个精化，从而保证编译过程中使用抽象策略的正确性。此精化关系可形式表示为一个TLA⁺中的蕴含关系

$$RTspec \Rightarrow Spec \tag{4-1}$$

根据本章对编译框架的设计，此蕴含关系很容易得证。为证明公式 (4-1)，首先将*RTspec*和*Spec*根据定义进行替换，从而得到

$$\begin{aligned} &BigInit \wedge \Box[BigNext]_{RTvars} \wedge RTL \\ &\quad \Rightarrow Init \wedge \Box[Next]_{vars} \wedge L \end{aligned} \tag{4-2}$$

为证明公式 (4-2)，只需证明以下三条公式即可

$$BigInit \Rightarrow Init$$

$$BigNext \Rightarrow (Next \vee \text{UNCHANGED } vars)$$

$$RTL \Rightarrow L$$

通过将*BigInit*，*BigNext*和*RTL*按照它们的定义进行替换，很容易分别证明这个合取公式的三个部分成立，从而公式 (4-1)成立。

因此，当验证需求P不涉及实时操作符时，我们在编译过程中将EDOLA模型翻译为抽象模型$Spec$，并在抽象模型$Spec$上验证属性P的成立性。根据已证明的精化关系，若公式$Spec$指定的行为满足属性P，即：$Spec \Rightarrow P$，则根据蕴含关系的传递性，$RTspec \Rightarrow P$ 成立，从而保证了抽象策略的正确性。基于抽象策略得到的subTLA模型*Spec* 与完整模型*RTspec*相比，包含更少的变量，因此减小了验证过程中的状态空间和搜索空间，从而为自动验证提供有效的支持。第 6 章的案例分析将验证这一结论。

4.4　subTLA到自动验证工具的转换

自动验证工具包含模型检测工具和自动定理证明工具两部分。TLC是以TLA⁺作为输入语言的模型检测工具，我们选用TLC作为EDOLA的模型检测

工具，从而省去了 subTLA到模型检测工具输入语言的转换工作。本节重点介绍subTLA到自动定理证明工具的转换方法。为表达方便，本章以自动定理证明工具SPASS[10]为例，介绍EDOLA的自动定理证明工作，但基本方法同样可应用于其它自动定理证明工具。

TLA逻辑中的已有推理规则，使得考虑subTLA到一阶逻辑自动定理证明工具的转换，并基于这些工具自动验证subTLA模型的性质成为可能。不考虑公平性约束，一个subTLA模型由范式$Init \wedge \Box[Next]_v$表示，验证一个归纳不变式（Inductive Invariant）P的TLA逻辑推理规则为[72]

$$\frac{Init \Rightarrow P,\ \ P \wedge Next \Rightarrow P',\ \ P \wedge v = v' \Rightarrow P'}{Init \wedge \Box[Next]_v \Rightarrow \Box P}. \tag{4-3}$$

即归纳不变式的证明被归结为三条动作层的假设。动作层的subTLA公式本质上是一阶逻辑上的公式，其中带撇变量v'（代表变量v在执行一个动作之后的值）被视为一个与不带撇变量v不同的变量。根据TLA$^+$语言中的已有验证成果，它的安全闭包（Safety Closure）性质[123]保证了TLA$^+$模型的公平性约束，不影响归纳不变式的证明结果，这使得我们可以将sub模型到SPASS的转换工作集中于subTLA模型动作定义。 TLA逻辑对于活性属性的证明，有类似的推理规则，但更为复杂，对此本章暂不考虑。

下面分别对subTLA到SPASS的基本转换方法、表达式的处理以及完整subTLA模块的处理进行介绍。

4.4.1 模型转换的基本知识

表达式是subTLA公式的基本组成部分，对它的有效处理是为subTLA模型提供证明支持的关键。在一阶逻辑中区分项（值）和公式（命题），而subTLA是无类型的。但本章的转换方法中包括了有限的一些类型推演，以保证转换后的模型结构良好，也为在转换过程中进行特定的优化提供了方便。

本章定义了两个递归函数 **T** 和 **S** 来分别表示类型推演和表达式转换，类型信息被表示为一个n维的函数或谓词：

$$(func, n, domL) \qquad 或 \qquad (pred, n, domL)$$

其中$func$表示函数，$pred$表示谓词；$n \geq 0$ 表示函数或谓词的维数（当$n = 0$ 时表

示常量）。当列表$domL$不为空（nil）时，表示一个n维的函数，$domL$中的每一成员记录了此函数在这一维的定义域。需要注意的是，$domL$存储的是对应元素转换之前的形式，即subTLA中的表达方式，这是因为这些元素在转换过程中可能会用于构造新的待转换subTLA表达式。我们在后面 EXCEPT 结构的处理中会看到这一点。

转换过程通过两遍扫描完成。第一遍扫描，应用函数 **T** 进行类型推演。给定一个语法结构，类型推演是一个自底向上的过程，即最底层的表达式最先被处理，高层表达式的类型信息由它的子表达式类型信息确定。第二遍扫描，基于已得类型信息，应用函数 **S** 进行转换。此过程是自顶向下的：函数 **S** 先作用于顶层表达式，再根据需要作用到它的子表达式或直接得到转换结果。

此外，我们还定义了函数 **Q**用于subTLA模型到一阶逻辑的正确转换。函数 **Q**只作用在标识符上，而不是一般的表达式。对于一个标识符id，$\mathbf{Q}[\![\, id \,]\!]$取值为真当且仅当 id 在表达式中作为一个（被全称或存在量词限定的）约束变量出现。对于一个基本的subTLA表达式 $x \in S$，根据S是否被量词所限定，有不同的转换规则。第 4.4.2.1 节将详细讲述这一点。

下面给出一个简单的例子来解释函数 **Q**。考虑表达式 $e_1 \subseteq e_2$，假设经过类型推演，可得知e_1和e_2的类型信息都为一元谓词（参考第 4.4.2.1 节)，即

$$\mathbf{T}[\![\, e_1 \,]\!] = \mathbf{T}[\![\, e_2 \,]\!] = (pred, 1, nil)$$

则转换过程中将得到

$$\begin{cases} \mathbf{S}[\![\, e_1 \subseteq e_2 \,]\!] = forall([x], implies(\mathbf{S}[\![\, x \in e_1 \,]\!], \mathbf{S}[\![\, x \in e_2 \,]\!])) \\ \mathbf{Q}[\![\, x \,]\!] = true \end{cases}$$

根据集合包含 $\subseteq$ 操作符的语义，容易理解此转换结果 ①。值得注意的是，由于新引入的变量x被全称量词所限定，因此它的 **Q** 函数值被设置为真。

4.4.2 表达式的转换方法

subTLA的表达式是TLA$^+$表达式的可自动验证子集。TLA$^+$中的记录和元组包含在subTLA的语法结构中，在翻译时作为一种特殊的函数考虑；集合乘积可

① SPASS中采用前缀操作符的形式表达逻辑运算，如蕴含操作符$implies$、逻辑与操作符and、逻辑或操作or、等式操作符$equal$、全称量词操作符$forall$和存在量词操作符$exists$等。

由元组和集合内涵进行定义，因此转换中不再直接考虑这些语法成分的转换。操作符ENABLED，UNCHANGED 操作符可被归结为本章所处理的结构，在此，不再给出它们的转换规则。本节将subTLA的表达式按照集合表达式、函数表达式、逻辑表达式、算术表达式以及其它表达式进行分类和介绍。

4.4.2.1　集合表达式

集合的编码规则　集合包含操作 $\in$ 是集合论中的原始算符。在一阶逻辑中，一般有两种方法对$x \in Set$进行编码

$$\mathbf{S}[\![\, x \in Set \,]\!] = \begin{cases} Set(x) & \text{即，将集合表示为一元谓词} \\ elem(x, Set) & \text{即，引入一个二元谓词} elem \end{cases}$$

多数一阶逻辑定理证明工具都针对一元谓词进行了优化。我们在SPASS上做了一些实验以检测这两种集合编码方法所带来的不同。将集合编码为一元谓词时，SPASS的证明效率提高很多，因此我们倾向于采用这种编码方式。然而，由于一阶逻辑的局限性，此编码方式不能适用于所有情形，例如，对于subTLA表达式 $\exists S \subseteq R : P$，其中 P 是一个谓词，若基于一元谓词对集合进行编码，会得到

$$\begin{aligned} & \mathbf{S}[\![\, \exists S \subseteq R : P \,]\!] \\ = {} & \mathbf{S}[\![\, \exists S : (S \subseteq R) \wedge P \,]\!] \\ = {} & exists([S], and(\mathbf{S}[\![\, S \subseteq R \,]\!], \mathbf{S}[\![\, P \,]\!]) \\ = {} & exists([S], and(forall([x], implies(\mathbf{S}[\![\, x \in S \,]\!], \mathbf{S}[\![\, x \in R \,]\!])), \mathbf{S}[\![\, P \,]\!])) \\ = {} & exists([S], and(forall([x], implies(S(x), R(x))), \mathbf{S}[\![\, P \,]\!])) \end{aligned}$$

从最后一行可以看出，翻译结果中包含了对谓词符号S的量化，所以不再是一个合法的一阶逻辑公式。这种情况下，我们采取第二种方法。也就是说，若 $\mathbf{Q}[\![\, S \,]\!] = true$，则将 $x \in S$ 转换为 $elem(x, S)$。基于此策略，上面的例子可转换为合法的一阶逻辑公式

$$exists([S], and(forall([x], implies(elem(x, S), R(x))), \mathbf{S}[\![\, P \,]\!]))$$

总体上讲，本章采用混成的方法来表示集合，即尽可能地将集合表示为一元谓词，并在一些特定的情况下，使用一个表示成员关系的二元谓词来显式表达集

合包含操作。

集合操作　接下来的内容介绍如何转换集合论中的一些经典操作。根据第 4.4.1 节介绍的基本转换知识，集合表达式 S 总是以 $e \in S$ 的形式出现。这里介绍subTLA中七种集合操作的类型推演规则以及到一阶逻辑的编码规则。

空集　$Exp ::= \{\}$　Exp被定义为一个空集，其类型推演和转换规则很简单：

$$\mathbf{T}[\![\, Exp \,]\!] = (pred, 1, nil), \qquad \mathbf{S}[\![\, e \in \{\} \,]\!] = false$$

集合枚举　$Exp ::= \{E_1, \ldots, E_n\}$　此表达式定义一个成员为 $E_1, \ldots, E_n$ 的枚举集合，其中$n \geq 1$。根据它的语义，可直观得到其类型推演和转换规则

$$\begin{aligned} \mathbf{T}[\![\, Exp \,]\!] &= (pred, 1, nil), \\ \mathbf{S}[\![\, e \in \{E_1, \ldots, E_n\} \,]\!] &= or(\mathbf{S}[\![\, e = E_1 \,]\!], \ldots, \mathbf{S}[\![\, e = E_n \,]\!]) \end{aligned}$$

基本的集合操作　基本的集合操作如 $\cup, \cap$和$\backslash$，根据它们的语义，可以直接转换为对应的布尔操作。例如

$$\mathbf{S}[\![\, e \in E_1 \setminus E_2 \,]\!] = and(\mathbf{S}[\![\, e \in E_1 \,]\!], not(\mathbf{S}[\![\, e \in E_2 \,]\!]))$$

集合约束　$Exp ::= \{Id \in E_1 : E_2\}$　此表达式定义了一个集合Exp，它是集合E_1的子集且集合成员必须满足谓词 E_2（其中Id是标识符）。类型推演和转换规则为

$$\begin{aligned} \mathbf{T}[\![\, Exp \,]\!] &= \mathbf{T}[\![\, E_1 \,]\!], \\ \mathbf{T}[\![\, E_2 \,]\!] &= (pred, 0, nil), \\ \mathbf{S}[\![\, e \in \{Id \in E_1 : E_2\} \,]\!] &= and(\mathbf{S}[\![\, e \in E_1 \,]\!], \mathbf{S}[\![\, E_2 \,|_{Id \leftarrow e} \,]\!]) \end{aligned}$$

其中$E_2 \,|_{Id \leftarrow e}$表示将$E_2$中所有出现$Id$的位置，用$e$替换。

集合内涵　$Exp ::= \{Exp_1 : Id_1 \in E_1, \ldots, Id_n \in E_n\}$　集合内涵通过指定它的成员必须满足的属性来描述一个集合。此集合内涵表达式定义了一个集合Exp，它由

所有满足下面条件的值 e 所构成：存在$Id_1 \in E_1, \ldots, Id_n \in E_n$，使得$e = Exp_1$。此转换规则定义为

$$\begin{aligned}
&\mathbf{S}[\![\, e \in \{Exp_1 : Id_1 \in E_1, \ldots, Id_n \in E_n\} \,]\!] \\
&\quad = \quad exists([Id_1, \ldots, Id_n], and(\mathbf{S}[\![\, Id_1 \in E_1 \,]\!], \ldots, \mathbf{S}[\![\, Id_n \in E_n \,]\!], \mathbf{S}[\![\, e = Exp_1 \,]\!])), \\
&\mathbf{Q}[\![\, Id_1 \,]\!] \quad = \quad true, \\
&\quad \ldots \\
&\mathbf{Q}[\![\, Id_n \,]\!] \quad = \quad true
\end{aligned}$$

幂集 Exp ::= **SUBSET** E_1　由于$e \in$ SUBSET E_1等价于$e \subseteq E_1$，因此转换规则定义为

$$\mathbf{S}[\![\, e \in \text{SUBSET } E_1 \,]\!] = \mathbf{S}[\![\, e \subseteq E_1 \,]\!]$$

广义并 Exp ::= **UNION** $\{Exp_1 : Id_1 \in E_1, \ldots, Id_n \in E_n\}$　TLA$^+$表达式 UNION S表示S 的所有成员集合的并集。若考虑此一般情形，转换规则将被定义为

$$\mathbf{S}[\![\, e \in \text{UNION } S \,]\!] = exists([X], and(\mathbf{S}[\![\, X \in S \,]\!], \mathbf{S}[\![\, e \in X \,]\!]))$$

然而，由于此规则包含了对幂集的直接表达，在使用SPASS进行证明时，效率较低下。因此，subTLA中表示的是一类特殊的 UNION 结构,对这类特殊的结构，可避免对幂集的直接表达，从而提高证明的效率。这类结构对应的转换规则定义为

$$\begin{aligned}
&\mathbf{S}[\![\, e \in \text{UNION } \{Exp_1 : Id_1 \in E_1, \ldots, Id_n \in E_n\} \,]\!] \\
= \quad &exists([Id_1, \ldots, Id_n], and(\mathbf{S}[\![\, Id_1 \in E_1 \,]\!], \ldots, \mathbf{S}[\![\, Id_n \in E_n \,]\!], \mathbf{S}[\![\, e \in Exp_1 \,]\!]))
\end{aligned}$$

4.4.2.2 函数表达式

函数在subTLA模型中起着至关重要的作用。实际应用中，函数常被用来表示程序设计语言中的数组等数据结构。此外，记录、元组等在subTLA中被视为特殊的函数，因此下面的转换规则对记录和元组同样适用。

函数表示　在subTLA中，函数 f 的定义域用 DOMAIN f 表示；定义域中的每个成员 x 被映射为值$f(x)$。本章使用SPASS中的函数来表示subTLA的函数，因此$\mathbf{T}[\![\, f \,]\!]$ = (func, n, domL)，其中 $n > 0$ 且 $domL$ 不为空。

另一种可能是将subTLA的函数编码为SPASS中的二元谓词如 f_p，使得 $f_p(x, y)$ 成立当且仅当 $x \in \text{DOMAIN}\, f$ 且 $y = f(x)$。函数的类型信息则被表示为：$\mathbf{T}[\![f]\!]$ = (pred, 2, nil)。然而，此编码的一个缺点是需要为每一函数定义一条公理，以声明函数值的唯一性。公理具有下面的形式：$forall([x, y_1, y_2], implies(and(f_p(x, y_1), f_p(x, y_2)), equal(y_1, y_2)))$。此公理对于表达函数的性质是必要的，若缺少此公理，很多定理将不能得证。另外，此编码规则在函数嵌套时会遇到麻烦。因此，本章采用第一种编码方式，即通过SPASS中的函数来表示subTLA中的函数，这种编码方法直观且便于定义转换规则。

表达式 $\text{DOMAIN}\, f$ 是表示函数 f 定义域的集合，因此 $\mathbf{T}[\![\text{DOMAIN}\, f]\!]$ = (pred, 1, nil)。给定函数 f 的类型信息 $\mathbf{T}[\![f]\!]$ = $(fund\ /\ pred, n, domL)$，$n > 0$ 且 $domL$ 不为空，则有：$\mathbf{S}[\![\text{DOMAIN}\, f]\!] = \mathbf{S}[\![Head(domL)]\!]$，其中 $Head(l)$ 返回列表 l 的第一个元素。

下面分别对subTLA中包含的函数定义以及操作的转换规则进行介绍。

函数定义 $Exp ::= [Id \in E_1 \mapsto E_2]$ **定**义了一个函数 f，定义域为 E_1，且对于任一 $Id \in E_1$，$f[Id]$ 的值等于 E_2（在subTLA中使用方括号表示函数作用）。考虑函数定义出现在等号表达式右侧的情形（其它情形可以归结为这种情形），转换规则定义为

$$\begin{aligned} & \mathbf{S}[\![e = [Id \in E_1 \mapsto E_2]]\!] \\ = \ & forall([x], implies(\mathbf{S}[\![x \in E_1]\!], \mathbf{S}[\![e[x] = E_2]\!])) \end{aligned}$$

且 $\mathbf{Q}[\![x]\!] = true$，$x$ 是转换过程生成的一个新量词变量。e和Exp的类型信息由 E_2 的类型信息决定

$$\mathbf{T}[\![e]\!] = \mathbf{T}[\![Exp]\!] = (pred/func, n + 1, E_1 :: domL),$$
$$\text{给定}\mathbf{T}[\![E_2]\!] = (pred/func, n, domL)$$

其中 :: 表示列表连接符。

函数作用 $Exp ::= E_1[e_1, \ldots, e_n]$ **表**示将函数 E_1 作用到 n 个参数 $e_1, \ldots, e_n$ 上。若函数E_1的维数m大于或等于n，则此函数作用被称为是结构良好的（Well-

Formed）。函数作用的转换规则定义为

$$\begin{aligned}&\mathbf{S}[\![\, E_1[e_1,\ldots,e_n]\,]\!] = \mathbf{S}[\![\, E_1\,]\!](\mathbf{S}[\![\, e_1\,]\!],\ldots,\mathbf{S}[\![\, e_n\,]\!])\\&\mathbf{T}[\![\, Exp\,]\!] = (pred/func, m-n, Tail(domL, n)),\\&\qquad\qquad 给定\mathbf{T}[\![\, E_1\,]\!] = (pred/func, m, domL)\end{aligned}$$

其中$m \geq n$，函数 $Tail(l, n)$ 返回列表l的尾（即删除前n个元素后得到的列表）。

函数重载 Exp ::= $[Exp_1$ EXCEPT $![E_1] = E]$ **构**造了一个函数Exp，它除了对参数 E_1 返回函数值 E 之外，和函数 Exp_1完全相同。给定Exp_1的类型信息 $\mathbf{T}[\![\, Exp_1\,]\!] = (pred/func, n, domL)$，函数重载操作的转换规则定义为

$$\begin{aligned}&\mathbf{S}[\![\, e = [Exp_1 \text{ EXCEPT } ![E_1] = E]\,]\!]\\&= forall([x], implies(\mathbf{S}[\![\, x \in Head(domL)\,]\!],\\&\qquad and(implies(\mathbf{S}[\![\, x = E_1\,]\!], \mathbf{S}[\![\, e[x] = E\,]\!]),\\&\qquad\quad implies(not(\mathbf{S}[\![\, x = E_1\,]\!]), \mathbf{S}[\![\, e[x] = Exp_1[x]\,]\!]))))\end{aligned}$$

表达式Exp的类型信息与Exp_1完全相同：

$$\mathbf{T}[\![\, Exp\,]\!] = \mathbf{T}[\![\, Exp_1\,]\!]$$

此转换很容易一般化到多函数成员更新的情形。subTLA中函数重载的更一般的形式为

$$[Exp_1 \text{ EXCEPT } ![e_{1,1}]\ldots[e_{1,n}] = E_1, \ldots, ![e_{k,1}]\ldots[e_{k,n}] = E_k]$$

函数空间 Exp ::= $[E_1 \rightarrow E_2]$ **表**示定义域为E_1且值域为E_2的函数集合。由于Exp是一个集合，因此本章同样考虑$e \in Exp$的转换规则

$$\begin{aligned}\mathbf{S}[\![\, e \in Exp\,]\!] &= forall([x], implies(\mathbf{S}[\![\, x \in E_1\,]\!], \mathbf{S}[\![\, e[x] \in E_2\,]\!]))\\\mathbf{Q}[\![\, x\,]\!] &= true\end{aligned}$$

函数相等 $Exp ::= e = E_1$ **表**达式e和E_1表示两个函数（可通过它们的类型信息进行判断）。函数相等的转换是递归进行的，一次转换函数的一维

$$\begin{aligned}\mathbf{S}[\![\, e = E_1 \,]\!] \;=\; & forall([x], implies(\mathbf{S}[\![\, x \in Head(domL) \,]\!], \mathbf{S}[\![\, e[x] = E_1[x] \,]\!])) \\ & \text{若 } \mathbf{T}[\![\, e \,]\!] = (pred/func, n, domL) \\ & \text{或 } \mathbf{T}[\![\, E_1 \,]\!] = (pred/func, n, domL)\text{，其中 } domL \text{ 不为 } nil\text{。}\end{aligned}$$

4.4.2.3 逻辑表达式

逻辑表达式到SPASS的转换比较直观，一些有代表性的逻辑表达式结构介绍如下。

集合BOOLEAN **在**subTLA中，BOOLEAN 是具有两个值TRUE和 FALSE的集合。这两个值可对应到SPASS中的值 $true$ 和 $false$，因此 BOOLEAN 可被编码为有两个成员的普通集合，从而对于表达式$p, q \in$ BOOLEAN，转换规则为（以p为例）：$\mathbf{S}[\![\, p \in \text{BOOLEAN} \,]\!] = \text{BOOLEAN}\,(p)$ 或 $\mathbf{S}[\![\, p \in \text{BOOLEAN} \,]\!] = elem(\text{BOOLEAN}, p)$，而$\mathbf{S}[\![\, p \wedge q \,]\!] = and(equal(p, true), equal(q, true))$。然而，它会给项替换（Term Substitution）带来更多可能性，因而在实践中效率很低。本章考虑了对 BOOLEAN 转换规则的优化：当遇到表达式$e \in$ BOOLEAN 时，我们将 e 识别为一个命题，并进行相应的处理。对于前面的例子，基于我们的转换方法，实际的转换结果为 $\mathbf{S}[\![\, p \in \text{BOOLEAN} \,]\!] = true$ 且 $\mathbf{S}[\![\, p \wedge q \,]\!] = and(p, q)$。

常量 $Exp ::= \text{TRUE} \mid \text{FALSE}$ **我**们仅考虑逻辑常量出现在等号表达式右部的情形（其它情形可以归结为这种情形），其转换规则和类型推演很容易理解

$$\begin{aligned}\mathbf{S}[\![\, e = \text{TRUE} \,]\!] &= \mathbf{S}[\![\, e \,]\!] \\ \mathbf{S}[\![\, e = \text{FALSE} \,]\!] &= not(\mathbf{S}[\![\, e \,]\!]) \\ \mathbf{T}[\![\, e \,]\!] = \mathbf{T}[\![\, Exp \,]\!] &= (pred, 0, nil)\end{aligned}$$

谓词等价 $Exp ::= e = Exp_1$ **已**知 e 或 Exp_1 的类型信息为：$(pred, n, nil)$，$n \geq 0$，转换函数 **S** 区分 $n = 0$ 和 $n > 0$ 两种情况对谓词等价操作进行转换

1. $\mathbf{S}[\![\, e = Exp_1 \,]\!] = equiv(\mathbf{S}[\![\, e \,]\!], \mathbf{S}[\![\, Exp_1 \,]\!])$
   若 $\mathbf{T}[\![\, e \,]\!] = (pred, 0, nil)$ 或 $\mathbf{T}[\![\, Exp_1 \,]\!] = (pred, 0, nil)$；

2. $\mathbf{S}[\![\, e = Exp_1 \,]\!] = forall([x_1, \ldots, x_n], equiv(\mathbf{S}[\![\, e \,]\!](x_1, \ldots, x_n), \mathbf{S}[\![\, Exp_1 \,]\!](x_1, \ldots, x_n)))$
且 $\mathbf{Q}[\![\, x_1 \,]\!] = \ldots = \mathbf{Q}[\![\, x_n \,]\!] = true$
若 $\mathbf{T}[\![\, e \,]\!] = (pred, n, nil)$ 或 $\mathbf{T}[\![\, Exp_1 \,]\!] = (pred, n, nil)$， $n \geq 1$。

有界量词 $Exp ::= \forall \mid \exists\, Id_1 \in E_1, \ldots, Id_n \in E_n : Exp_1$ 一阶逻辑中有（无界的）全称和存在量词，因此只需将它们与有界集合按照第 4.4.2.1 节给出的集合结构转换规则进行关联即可。以有界全称量词为例，转换规则和类型推演定义为

$$
\begin{aligned}
&\mathbf{S}[\![\, \forall Id_1 \in E_1, \ldots, Id_n \in E_n : Exp_1 \,]\!] = \\
&forall([Id_1, \ldots, Id_n], implies(and(\mathbf{S}[\![\, Id_1 \in E_1 \,]\!], \ldots, \mathbf{S}[\![\, Id_n \in E_n \,]\!]), \mathbf{S}[\![\, Exp_1 \,]\!])), \\
&\mathbf{Q}[\![\, Id_1 \,]\!] = true \\
&\quad \ldots \\
&\mathbf{Q}[\![\, Id_n \,]\!] = true \\
&\mathbf{T}[\![\, Exp \,]\!] = (pred, 0, nil), \qquad \mathbf{T}[\![\, Exp_1 \,]\!] = (pred, 0, nil)
\end{aligned}
$$

4.4.2.4 算术运算表达式

算术运算表达式一般涉及到自然数、整数和实数的运算。虽然一阶逻辑具有足够强的表达能力来形式化（一阶）Peano算术理论，但从效率上来讲，目前一阶逻辑自动证明工具处理算术问题还有一定的困难。近年来，随着可满足性模理论SMT 的发展，一些流行的一阶逻辑定理证明工具如SPASS、CVC3等都在某种程度上提供了对线性代数理论的支持。对于这类自动定理证明工具，subTLA的线性运算操作可以进行相应的转换。譬如，SPASS(LA)是在基本SPASS上加入线性代数理论的版本，subTLA中的算术操作如≥、≤、+、∗等可直接对应到SPASS(LA)中的相应操作符。其转换规则只涉及到语法表示的改变，例如

$$
\begin{aligned}
\mathbf{S}[\![\, e_1 \geq e_2 \,]\!] &= ge(\mathbf{S}[\![\, e_1 \,]\!], \mathbf{S}[\![\, e_2 \,]\!]) \\
\mathbf{S}[\![\, e_1 \leq e_1 \,]\!] &= ls(\mathbf{S}[\![\, e_1 \,]\!], \mathbf{S}[\![\, e_2 \,]\!]) \\
\mathbf{S}[\![\, e_1 + e_2 \,]\!] &= \mathbf{S}[\![\, e_1 \,]\!] + \mathbf{S}[\![\, e_2 \,]\!] \\
\mathbf{S}[\![\, e_1 * e_2 \,]\!] &= \mathbf{S}[\![\, e_1 \,]\!] * \mathbf{S}[\![\, e_2 \,]\!]
\end{aligned}
$$

4.4.2.5　特殊表达式

subTLA中的一些表达式在一阶逻辑中没有直接的对应成分，这些典型结构介绍如下。

条件表达式 Exp ::= IF e_1 THEN e_2 ELSE e_3　条件表达式转换被分布到对包含它的表达式的处理，直到条件表达式出现在语法结构的某一层，使得它们可被转换为蕴含关系。这里考虑的是*if*表达式出现在中缀操作符右端的情形，转换规则定义为

$$
\begin{aligned}
&\mathbf{S}[\![\ e\ infixOp\ \text{IF}\ e_1\ \text{THEN}\ e_2\ \text{ELSE}\ \ e_3\ ]\!] = \\
&and(implies(\mathbf{S}[\![\ e_1\ ]\!], \mathbf{S}[\![\ e\ infixOp\ e_2\ ]\!]), \\
&\qquad implies(not(\mathbf{S}[\![\ e_1\ ]\!]), \mathbf{S}[\![\ e\ infixOp\ e_3\ ]\!])) \\
&\mathbf{T}[\![\ e_1\ ]\!] = (pred, 0, nil) \\
&\mathbf{T}[\![\ Exp\ ]\!] = \mathbf{T}[\![\ e_2\ ]\!] = \mathbf{T}[\![\ e_3\ ]\!]
\end{aligned}
$$

其中$infixOp \in \{\wedge, \vee, \Rightarrow, \equiv, \cap, \cup, \setminus, \in, \notin, =, \neq, \subseteq \geq, \geq, \leq, \leq, +, -, *, /\}$，是subTLA的一个中缀操作符。

Case表达式 Exp ::= CASE $e_1 \rightarrow E_1 \ \square \ldots \square\ e_n \rightarrow E_n$ ($\square$ OTHER $\rightarrow E_{n+1}$)?　Case表达式将条件表达式一般化为具有多个分支的情形，它的转换规则与IF 表达式类似，同样考虑其出现在中缀表达式右部的情形。以包含OTHER 分支的*case*表达式为例，其转换规则定义为

$$
\begin{aligned}
&\mathbf{S}[\![\ e\ infixOp\ \text{CASE}\ e_1 \rightarrow E_1 \ \square \ldots \square\ e_n \rightarrow E_n\ \square\ \text{OTHER}\ \rightarrow E_{n+1}\ ]\!] = \\
&and(implies(\mathbf{S}[\![\ e_1\ ]\!], \mathbf{S}[\![\ e\ infixOp\ E_1\ ]\!]), \\
&\qquad \ldots, \\
&\qquad implies(\mathbf{S}[\![\ e_n\ ]\!], \mathbf{S}[\![\ e\ infixOp\ E_n\ ]\!]), \\
&\qquad implies(and(not(\mathbf{S}[\![\ e_1\ ]\!]), \ldots, not(\mathbf{S}[\![\ e_n\ ]\!])), \mathbf{S}[\![\ e\ infixOp\ E_{n+1}\ ]\!])) \\
&\mathbf{T}[\![\ e_1\ ]\!] = \ldots = \mathbf{T}[\![\ e_n\ ]\!] = (pred, 0, nil) \\
&\mathbf{T}[\![\ Exp\ ]\!] = \mathbf{T}[\![\ E_1\ ]\!] = \ldots = \mathbf{T}[\![\ E_{n+1}\ ]\!]
\end{aligned}
$$

其中 $infixOp$ 是subTLA的中缀操作符，与前面的定义相同。

带撇变量 $Exp ::= Id'$ **在**subTLA动作公式中，状态变量以带撇和不带撇两种形式出现，其中带撇变量v'表示变量v在执行一个动作之后的值，而v则表示执行此动作之前的值。在转换中，v和v'被识别为一阶逻辑中的两个不同的变量或命题。

CHOOSE表达式 完整TLA$^+$语言的 CHOOSE 操作符有两种用法，无界的形式：CHOOSE $Id : E_1$，或有界的形式：CHOOSE $Id \in E_1 : E_2$，而后者可由无界的形式定义出来，即CHOOSE $Id \in E_1 : E_2 \triangleq$ CHOOSE $Id : (Id \in E_1) \wedge E_2$，因此转换时仅考虑无界的形式即可。

根据CHOOSE 操作符在TLA$^+$中的语义，若命题 E_1 是可满足的，则表达式CHOOSE $Id : E_1$ 的值是任一使得命题E_1 成立的值v（当把E_1中所有Id的出现都替换为v时）；否则（即命题E_1不可满足），此表达式将返回一完全任意的值[26]。若根据此语义，CHOOSE 表达式的类型推演和转换规则可定义为

$$\begin{aligned} \mathbf{T}[\![\ \text{CHOOSE}\ Id : E_1\ ]\!] &= \mathbf{T}[\![\ Id\ ]\!] \\ \mathbf{T}[\![\ E_1\ ]\!] &= (pred, 0, nil) \\ \mathbf{S}[\![\ \text{CHOOSE}\ Id : E_1\ ]\!] &= Idnew \end{aligned}$$

并伴随着一条公理

$$\begin{aligned} &formula(and(implies(exists([Id], \mathbf{S}[\![\ E_1\ ]\!]), \mathbf{S}[\![\ E_1|_{Id \leftarrow Idnew}\ ]\!]), \\ &\qquad implies(not(exists([Id], \mathbf{S}[\![\ E_1\ ]\!]), \mathbf{S}[\![\ Idnew = ARBIT\ ]\!])))). \end{aligned}$$

其中$Idnew$是一个新生成的标识符，$ARBIT$ 是一个预定义的常量，类型信息为$\mathbf{T}[\![\ ARBIT\ ]\!]$=(func, 0,nil)，表示当命题$E_1$不可满足时，表达式将返回的那个任意值。

然而，SPASS却不能基于这个一般形式的CHOOSE 表达式转换规则进行预期的推理。这是因为SPASS无法判定上述两个互斥条件中，应该选择哪一个。因此，subTLA中仅考虑CHOOSE 表达式的一些特殊情形，在这些情况下，可根据已有知识可得知互斥条件中哪一个成立，从而将转换规则归结到其中一条分支。subTLA中包含的特殊CHOOSE 表达式及其转换规则描述如下。

1. Exp::= CHOOSE Id : FALSE. 由于FALSE是不可满足的，所以第二条分支被选择。转换规则为：$\mathbf{S}[\![\ Exp\ ]\!] = ARBIT$；

2. Exp::= CHOOSE $Id : Id \in \{\}$. 由于空集中不包含任何元素，因此第二条分支被选择，转换规则为：$\mathbf{S}[\![Exp]\!] = ARBIT$；
3. Exp::= CHOOSE $Id : Id \in E_1$，其中 E_1 非空。在这种情况下，存在变量Id，使得Id属于集合E_1，因此第一条分支被选择，转换规则定义为：$\mathbf{S}[\![Exp]\!] = Idc$，并伴随着一条公理：*formula*($\mathbf{S}[\![Idc \in E_1|_{Id \leftarrow Idc}]\!]$). 若此CHOOSE表达式是第一次在subTLA模型中出现，则Idc表示一个新生成的标识符；否则 Idc表示对其第一次出现时所定义标识符的引用。
4. Exp::= CHOOSE $Id : Id \notin E_1$. 由于在 Zermelo-Fränkel集合论中，不存在“包含了所有集合的集合”。因此任给一个集合，总是存在不属于这个集合的元素，从而使得$Id \notin E_1$可满足，所以第一条分支被选择，转换规则定义为：$\mathbf{S}[\![Exp]\!] = Idc$，伴随着公理：*formula*($\mathbf{S}[\![Idc \notin E_1|_{Id \leftarrow Idc}]\!]$)., 其中$Idc$的解释和前面相同。

4.4.3 完整subTLA模型的处理方法

一个完整的subTLA模型由声明和定义两部分组成。在声明部分声明模型用到的所有常量和变量；在定义部分，各个动作以及相关概念以公式的形式进行定义。

subTLA是无类型的，常量和变量声明部分不包含任何类型信息。因此，转换过程中所需的类型信息，通过在使用常量或变量时，对其进行类型推演来确定。例如，一个subTLA模型中通过语句 CONSTANTS $Clients$ 声明了一个标识符$Clients$；我们将其记录下来，但尚不能猜测关于$Clients$的任何类型信息。之后，通过对整个模型的扫描，我们为$Clients$推演出它的类型信息，例如$(pred, 1, nil)$，进而在SPASS中将其声明为一个一元谓词，形式为 $predicates[(Clients, 1), \ldots]$。

subTLA的动作以及相关概念的公式定义具有形式 $LeftF \triangleq Exp$，其中$LeftF \triangleq Id \,|\, Id(Id_1, \ldots, Id_n)$。一条subTLA公式的转换，通过对其右部表达式进行转换，并在SPASS中添加相应的声明来实现。具体来说，公式$LeftF \triangleq Exp$的转换，对应于SPASS的一条公式

$$formula(\mathbf{S}[\![LeftF = Exp]\!]).$$

它的转换被归结为对表达式$LeftF = Exp$的转换，公式左部的标识符要添加

到SPASS 输入文件的符号声明部分。例如，对于一个定义$available \triangleq Resources \setminus allocated$，按照本章定义的规则对其进行翻译：在类型推演过程中，可得知$available$ 的类型信息，如$\mathbf{T}[\![available]\!] = (pred, 1, nil)$，进而，我们需在SPASS输入文件的声明部分添加对一元谓词$available$的声明。

4.5　本章小结

本章介绍了验证支持层的设计方法和实践。提出基于中间语言的转换方法，实现了多个EDOLA的验证支持重用。中间语言选用形式语言TLA$^+$的子集subTLA，保证了自动验证性。在EDOLA到subTLA的编译过程中，提出面向验证需求的抽象策略，以减少验证过程中的状态空间和搜索空间。在subTLA的自动定理证明方法中，制定了subTLA关键语法成分：集合、函数等表达式到一阶逻辑的有效编码规则，使得转换后的subTLA模型能够被SPASS等自动定理证明工具有效处理，从而借助已有工具实现了对无限空间模型的自动验证。

第 5 章 EDOLA-PLC的原型设计和工具实现

根据前几章的研究成果，本章提出一种EDOLA语言的体系结构，并基于此体系结构构造了PLC领域建模验证语言EDOLA-PLC的原型。为便于语言的应用，开发了EDOLA-PLC 的工具，包括语言的编辑器和编译器。编辑器提供了基本的用户操作界面；编译器实现了语言的语法、语义检查以及基于转换的自动验证等功能。

5.1 引言

本文第 1 章提出了EDOLA的三层设计结构，第 2 到 4 章分别介绍了领域知识层、公共模块层和验证支持层的研究和实践。前几章的局部研究成果将通过EDOLA语言设计进行综合。本章介绍面向PLC控制软件领域的建模验证语言EDOLA-PLC的原型设计和工具实现。EDOLA-PLC的原型设计遵循领域知识层、公共模块层和验证支持层的三层结构，综合实现了前几章的研究成果。

本章首先提出EDOLA语言的一种体系结构，实现EDOLA设计的三层结构在语言语法上的对应表示。基于此体系结构，我们开发了EDOLA-PLC语言，本章将给出其抽象语法，并对语法元素与三层结构的对应关系进行介绍。为了简洁，本章略去了EDOLA-PLC语义的形式定义，这部分内容将通过第 6 章的码头消防控制实例进行解释。

EDOLA-PLC语言的应用实践离不开工具的支持。我们实现了EDOLA-PLC语言的工具原型，包括语言的编辑器和编译器两部分，以提供与用户交互的接口、语法和语义分析以及自动验证等功能。

本章的其余部分组织如下：第 5.2 节介绍EDOLA的体系结构；第 5.3 节介绍EDOLA-PLC的抽象语法；第 5.4 节介绍EDOLA-PLC的工具原型实现；第 5.5 节是对本章的小结。

5.2 EDOLA的体系结构

前几章分别介绍了EDOLA的三层设计结构以及各层的设计方法和实践。本

节给出 EDOLA的体系结构，以将EDOLA的设计思想和研究成果实现在语言的语法成分中，如图 5.1 所示。EDOLA语言由模块构成，扩展模块声明部分可将

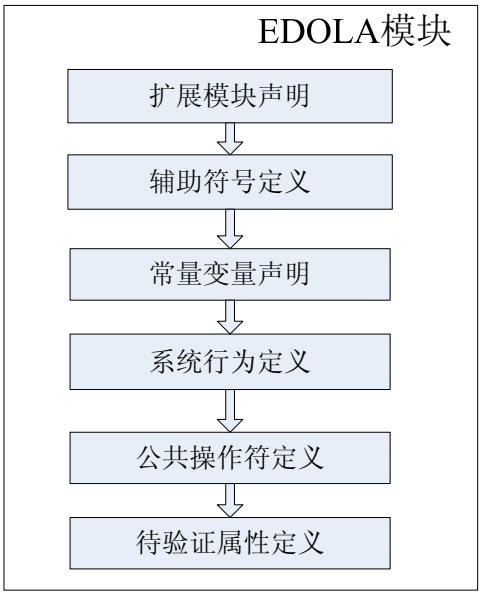

图 5.1　EDOLA语言的体系结构

其它EDOLA模块包含进来，以便为较大规模的系统描述提供模块化支持。辅助符号定义部分用来定义为方便建模而引入的一些辅助符号，如集合定义、局部动作定义等。常量变量声明部分对模型的参数以及模型的变量和类型进行声明。系统行为定义部分描述EDOLA模型的动态行为，即模型是如何演进的，这里基于状态变迁系统形式进行定义，分为初始状态定义和下一步动作定义两部分。公共操作符定义部分针对不同领域，直接引用已定义的公共操作符或将其进一步封装为新的EDOLA语法成分。最后一部分是待验证属性定义，通过应用属性操作符，描述期望验证的属性。

EDOLA语言体系结构中，领域知识层的研究成果应用于常量变量声明、系统行为定义以及待验证属性语法符号的设计。公共模块层的研究成果应用于公共操作符定义部分的语法符号设计。验证支持层的研究成果不通过EDOLA的语法，而是通过语义进行体现。

5.3 EDOLA-PLC的抽象语法

本节介绍EDOLA-PLC的抽象语法以及各语法元素与三层结构研究成果的对应关系。EDOLA-PLC的抽象语法由BNF表达，如图 5.2 和图 5.3 所示。为了简洁，图中略去了表达式的具体语法定义。EDOLA-PLC表达式的定义从subTLA表达式定义继承而来，见第 4.4.2 节的描述。EDOLA-PLC的形式语义通过定义EDOLA-PLC模型所对应的subTLA模型给出。由于部分形式语义已在第 2 章介绍，为避免重复以及更直观的理解，此处略去 EDOLA-PLC严格的形式定义，而是在第 6 章通过码头消防实例进行介绍。

EDOLA-PLC语言由模块构成，模块的定义格式参考了TLA$^+$的模块。模块定义的开始，通过前后至少四个连续的“-”字符、关键字 **MODULE** 和模块名 $ModuleName$ 组合表示；模块定义的结束，由至少四个连续的“=”字符表示。EDOLA-PLC模块的语法成分构成，按先后顺序分别为：模块扩展、辅助定义、声明、动作定义、约束和待验证属性。

模块扩展通过 **EXTENDS** 关键字引出，其后跟上一个模块名或由逗号分隔开的一组模块名。EDOLA-PLC的模块扩展表示的含义为，“**EXTENDS** M” 语句将模块 M 中的所有内容添加到包含这条语句的模块中。因此，模块 M 中的声明、定义、约束和待验证属性以及它所扩展的所有其它模块都被包含进来。

辅助定义（$GeneralDef$）由公式定义序列构成。为了使变量声明和动作定义更清晰，用户可在此部分定义一些函数、谓词、局部动作等，以便在之后的声明或定义中对其进行引用。

声明部分（$Declarations$）包括常量声明和变量声明。常量声明通过关键字 **CONSTANT** 引出。EDOLA-PLC的常量可以是由一个标识符表示的常量，或由标识符、左括号、逗号隔开的“_”字符以及右括号组合表示的常函数，其中标识符表示常函数的名字，“_”字符的个数表示函数的维数。变量声明分为输入变量声明、系统变量声明和输出变量声明三部分。输入变量声明由关键字 **INPUTVAR** 引出，表示PLC的环境输入变量（如用户的控制台命令输入和设备反馈信号），输入变量值在PLC的一个循环周期中保持不变。输出变量声明由关键字 **OUTPUTVAR** 引出，表示PLC控制软件对于环境的输出，用于控制设备的启停或在控制面板上显示系统的状态。系统变量声明由关键字 **SYSTEMVAR** 引出，表示实现PLC 控制功能时引入的辅助变量，它们的值在系统行为的动作定

```
EDOLA-module ::=      AtLeast4("-")  MODULE  ModuleName  (AtLeast4("-")
                      ( nil  |  EXTENDS  CommaList( ModuleName) )
                      GeneralDef
                      Declarations
                      ActionDef
                      Constraints
                      Properties
                      AtLeast4("=")
GeneralDef ::= nil  |  ( formula )*
Formula ::= LeftF  '= ='  Exp
LeftF ::= Name  |  Name  "("  CommaList(ID)  ")"
Declarations ::=      ConstDeclarations     VarDeclarations
ConstDeclarations ::= CONSTANT CommaList ( OpDec )
OpDec ::= ConstName  |  ConstName  "("  CommaList("_")  ")"
VarDeclarations ::=   INPUTVAR  varDecList
                      OUTPUTVAR  varDecList
                      SYSTEMVAR  varDecList
varDecList ::= CommaList ( varTypeDes )
varTypeDes ::= varName \in Exp
ActionDef ::=      INIT  formula
                   ACTION  ActionList
                   SPEC  SpecName
ActionList ::=  ( Formula )+
Constraints ::=       EnvConstraint
                    ( nil | TimeConstraints )
EnvConstraint ::=    ENV  TOTAL
                   |  ENV  ( Formula )+
TimeConstraints ::= TIME ( Duration | Interval | Delay | Deadline | Timeout | WaitUntil ) +
Duration ::= (Quantif)? DURATION
              BOUND (STRFlag, TimerName, ActRef, minVal, LowerFlag, maxVal, UpperFlag)
           |  UB (STRFlag, TimerName, ActRef, maxVal, UpperFlag)
           |  LB (TimerName, ActRef, minVal, LowerFlag)
           |  VALUE (STRFlag, TimerName, ActRef, Val)
Interval ::= (Quantif)? INTERVAL
              BOUND (STRFlag, TimerName, ActRef, ActRef, minVal, LowerFlag, maxVal, UpperFlag)
           |  UB (STRFlag, TimerName, ActRef, ActRef, maxVal, UpperFlag)
           |  LB (TimerName, ActRef, ActRef, minVal, LowerFlag)
           |  VALUE (STRFlag, TimerName, ActRef, ActRef, Val)
```

图 5.2 EDOLA-PLC的抽象语法

Delay ::= (Quantif)? **DELAY** (ActRef, Val, TimerName)
Deadline ::= (Quantif)? **DEADLINE** (ActRef, Val, TimerName)
Timeout ::= (Quantif)? **TIMEOUT** (ActRef, ActRef, Val, TimerName, TimerName)
WaitUntil ::= (Quantif)? **WAITUNTIL** (ActRef, IdleName, Val)
Quantif ::= (\A | \E) CommaList (Name \in SetName) ":"
ActRef ::= ActName | ActName "(" CommaList(ID) ")"
Properties ::= **PROP** (Debug | ToProof)
Debug ::= **DEBUG** PropName: (Quantif)? ActRef
ToProof ::= **TOPROOF** (Respond | Compete | Sequence | Priority | StateInv | ActInv)*
Respond ::= PropName ":" (Quantif)? **RESPOND** (ActRef, EnvState, SysState)
Compete ::= PropName ":" (Quantif)? **COMPETE** (CondExp, SysState, SysState)
Sequence ::= PropName ":" (Quantif)? **SEQUENCE** (ActRef, sysState)
Priority ::= PropName ":" (Quantif)? **PRIORITY** (ActRef , SysState)
StateInv ::= PropName ":" (Quantif)? **STATEINV** (SysState)
ActInv ::= PropName ":" (Quantif)? **ACTINV** (ActRef)
EnvState ::= Exp
SysState ::= Exp
STRFlag ::= TRUE | FALSE
LowerFlag ::= TRUE | FALSE
UpperFlag ::= TRUE | FALSE
ModuleName ::= ID
Name ::= ID
ActName ::= ID
SetName ::= ID
TimerName ::= ID
IdleName ::= ID
Val ::= DIGIT
minVal ::= DIGIT
maxVal ::= DIGIT

图 5.3 EDOLA-PLC的抽象语法（续）

义中通常会发生改变。

动作定义由初始状态、动作序列和规范说明三部分构成。初始状态定义由关键字 **INIT** 引出的一条公式表示，为所有的变量赋初始值。动作序列由关键字**ACTION**引出的公式定义序列提供，每条公式定义PLC控制软件在当前系统状态（由所有变量的当前值表示）下的一个动作。规范说明则表达为关键字 **SPEC** 引出的一个标识符，表示在验证阶段引用EDOLA-PLC模型时使用的名字。

约束部分包括环境约束和实时约束，实时约束是可选的。环境的表示分为两种情况：高可信要求下的全环境模型由关键字 **ENV DEFAULT** 给出，表示环境输入可以是任意值；而用户自定义环境描述则由关键字 **ENV** 开头的一组公式定义构成。实时约束部分由关键字**TIME**引出，基于第 3 章实时公共操作符的定义，一个实时约束可以是表示单个动作的时间约束*Duration*、表示动作间隔的时间约束*Interval*、高级时间操作符*Delay*、*Deadline* 、*Timeout*或*WaitUntil*。各操作符由对应的关键字（如**DURATION BOUND**）和一组参数表示。此外，EDOLA-PLC允许定义带参数的动作，因此动作引用由 $ActRef$ 表示，可以是一个由标识符表示的动作名，或由标识符、括号和参数构成的带参动作。当 $ActRef$ 为带参动作时，由 $Quantif$ 给出参数的取值范围。各个参数与它们在第 3 章中的解释完全相同，唯一的区别在于，EDOLA-PLC描述中不再需要给出参数v，它将由EDOLA-PLC的编译器自动生成。

属性部分由关键字 **PROP** 引出，根据第 2 章定义的属性操作符，这部分又分为排错属性操作符定义和正确性属性操作符定义两类。排错属性由关键字**DEBUG** 给出的一个动作引用表示，用于测试该动作是否允许发生，$PropName$定义这条属性的名字。正确性属性由 **TOPROOF** 关键字引出，可以是由 **RESPOND** 关键字表示的响应性质、 **COMPETE**关键字表示的竞争性质、**SEQUENCE** 关键字表示的顺序性质、**PRIORITY**关键字表示的优先性质、**STATEINV** 表示的状态不变式、或 **ACTINV** 关键字表示的动作不变式。

EDOLA-PLC的语法设计融合了领域知识层和公共模块层的研究成果。变量声明部分的三种变量分类、约束部分的两类环境区分等实现了PLC的领域知识描述；属性描述的各种操作符定义实现了PLC领域验证需求的描述；实时约束部分的各种时间操作符定义实现了实时公共模块层的研究成果。EDOLA-PLC的自动验证特征在它到subTLA的转换中体现，我们将在第 6 章通过案例进行介绍。

5.4　EDOLA-PLC的工具实现

为便于用户使用，我们初步实现了EDOLA-PLC的工具，包括EDOLA-PLC的编辑器和编译器。EDOLA-PLC的编辑器提供与用户交互的接口，编译器部分负责模型的检查、转换和验证。

5.4.1　EDOLA-PLC的编辑界面和功能介绍

编辑界面基于Netbeans IDE 6.7实现。初步的EDOLA-PLC用户界面如图 5.4 所示。 编辑器的主窗口包含两个部分，上部窗口用于编辑和显示文本形式

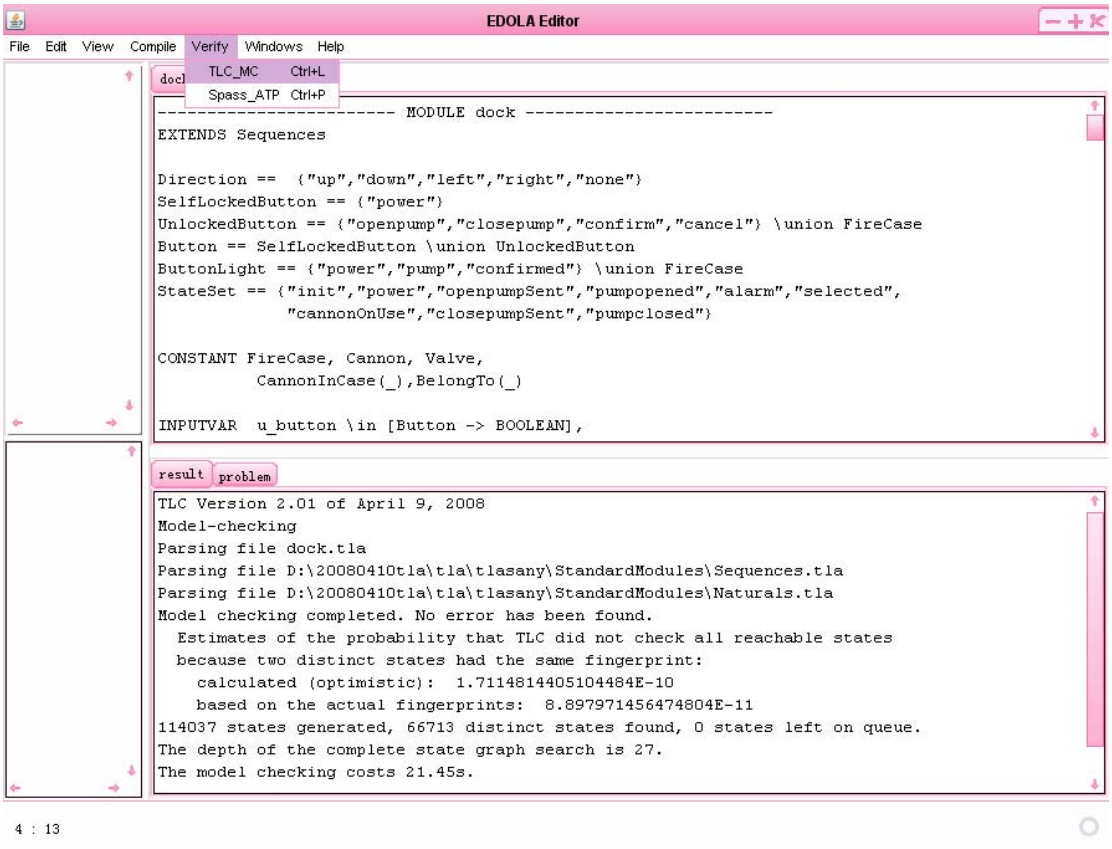

图 5.4　EDOLA-PLC的用户界面

的EDOLA-PLC模型，下部窗口用于显示各种执行结果和错误信息。编辑器的基本功能通过菜单项来实现。菜单可以分为基本功能部分和语言功能部分。

与常见编辑器类似，基本功能部分实现了文件（File）、编辑（Edit）和视图（View）菜单。文件菜单包括新建（New）、打开（Open）、保存（Save）、另存（Save as）、打印（Print）等基本功能。用户选择Open菜单项时，编辑器供用户选择并加载一个以.edo结尾的EDOLA-PLC 模型文件；用户选择New菜单项时，工具自动加载一个.edo文件模板，模板中提供了EDOLA-PLC模型必要的语法成分（如模块开始、模块结束、变量声明关键字等）以及一些注释信息，以辅助用户编写EDOLA-PLC模型。编辑菜单实现了基本的拷贝（Copy）、粘贴（Paste）、重做（Redo）和回退（Undo）等基本功能；视图菜单实现了字体设置、颜色设置、放大、缩小等功能，以便于EDOLA-PLC模型显示。

在语言功能部分，提供了编译（Compile）菜单和验证（Verification）菜单。编译菜单用于单步执行，而验证菜单则用于模型验证。在编译菜单中可通过语法分析（Syntax analysis）、语义分析（Semantic analysis）、subTLA模型生成（subTLA generation）、基于TLC的subTLA模型检测（subTLA TLC-MC）、SPASS输入文件生成（Spass input）以及基于SPASS的模型验证等菜单项来单步执行 EDOLA-PLC的模型验证，结果和错误信息在下部窗口显示。验证菜单提供了TLC模型检测（TLC-MC）和SPASS（Spass-ATP）自动定理证明两种验证选择，实现了验证中间过程的无缝链接，因此可直接对.edo模型进行自动验证，验证结果也在下部窗口显示。

5.4.2 EDOLA-PLC的编译器

EDOLA-PLC编译器的实现架构如图 5.5 所示。编译器的用户输入为.edo文件描述的EDOLA-PLC模型；编译器首先将EDOLA-PLC模型转换为一个subTLA模型，通过.tla文件表示。转换过程由三个步骤实现：语法检查、语义检查和转换。若模型存在语法、语义错误或转换错误，则提供相应的错误提示信息，由.err文件表示；否则生成一个 subTLA模型。subTLA模型将根据用户的验证选择，与subTLA配置文件（由.cfg表示）组合，提供给工具TLC 进行模型检测，或生成SPASS模型（由.dfg表示）提供给工具SPASS进行自动定理证明。模型检测的验证结果由.tlr表示，自动定理证明的结果由.spr表示。与用户界面菜单项结合，当单步执行时（选择编译菜单的菜单项），将以.err文件、.tla文件 .dfg文件、.tlr文件或.spr的文件的内容作为输出，显示在下部窗口；当模型验证时（选

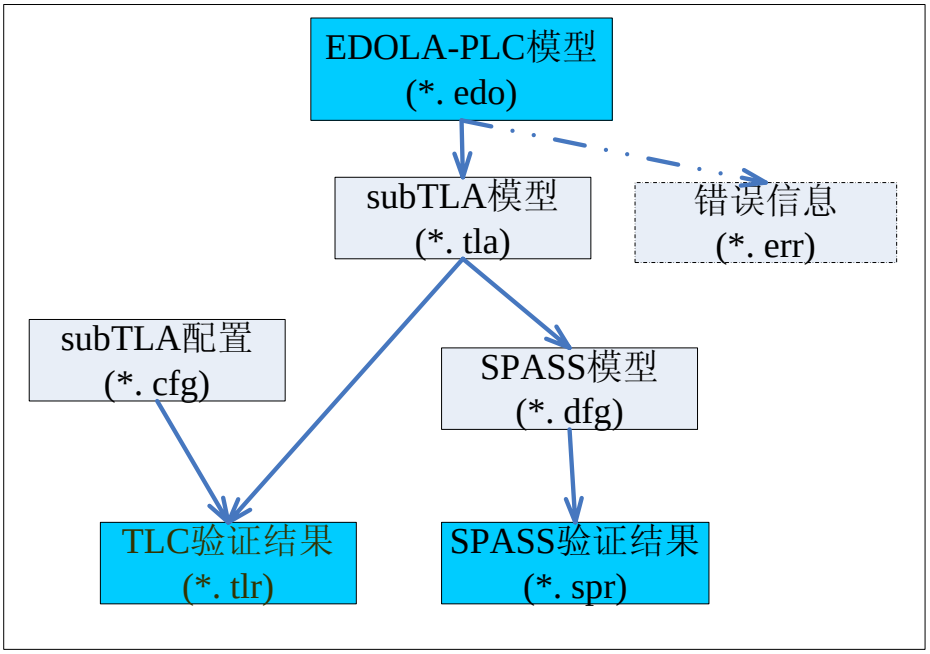

图 5.5　EDOLA-PLC的编译器实现架构

择验证菜单的菜单项），则仅以.err、.tlr文件或.spr文件的内容作为输出。

EDOLA-PLC的语法检查通过语法分析器构造工具 JavaCC[124]（Java Compiler Compiler）实现。目前工具中已实现的语义检查包括：

- 变量未定义；
- 变量未初始化；
- 带参动作定义中未指定参数取值范围；
- 动作定义中的变量重复赋值；
- 输入变量值在系统动作定义中被修改（警告）；
- 环境定义中修改了系统变量值（警告）。

EDOLA-PLC 的语义检查、EDOLA-PLC 到 subTLA 的转换以及 subTLA 到 SPASS 模型的转换都是基于 JavaCC 以语法制导的方式实现的。 JavaCC是一个广泛使用的LL算法分析工具，它接收正则表达式和BNF符号提供的词法和文法输入，并自动生成Java代码实现的词法语法分析器。我们使用的工具版本为JavaCC 4.2，它提供了JJTree等预处理工具以便在语法分析过程构造语法树。EDOLA-PLC编译器的实现语言为JAVA1.6，实现平台为Windows操作系统下的Eclipse3.4.1 IDE工具。下面以subTLA到SPASS的转换为例，介绍基于JavaCC的

模型转换实现。基于JavaCC的subTLA到SPASS模型转换所涉及的文件及其关系如图 5.6 所示。

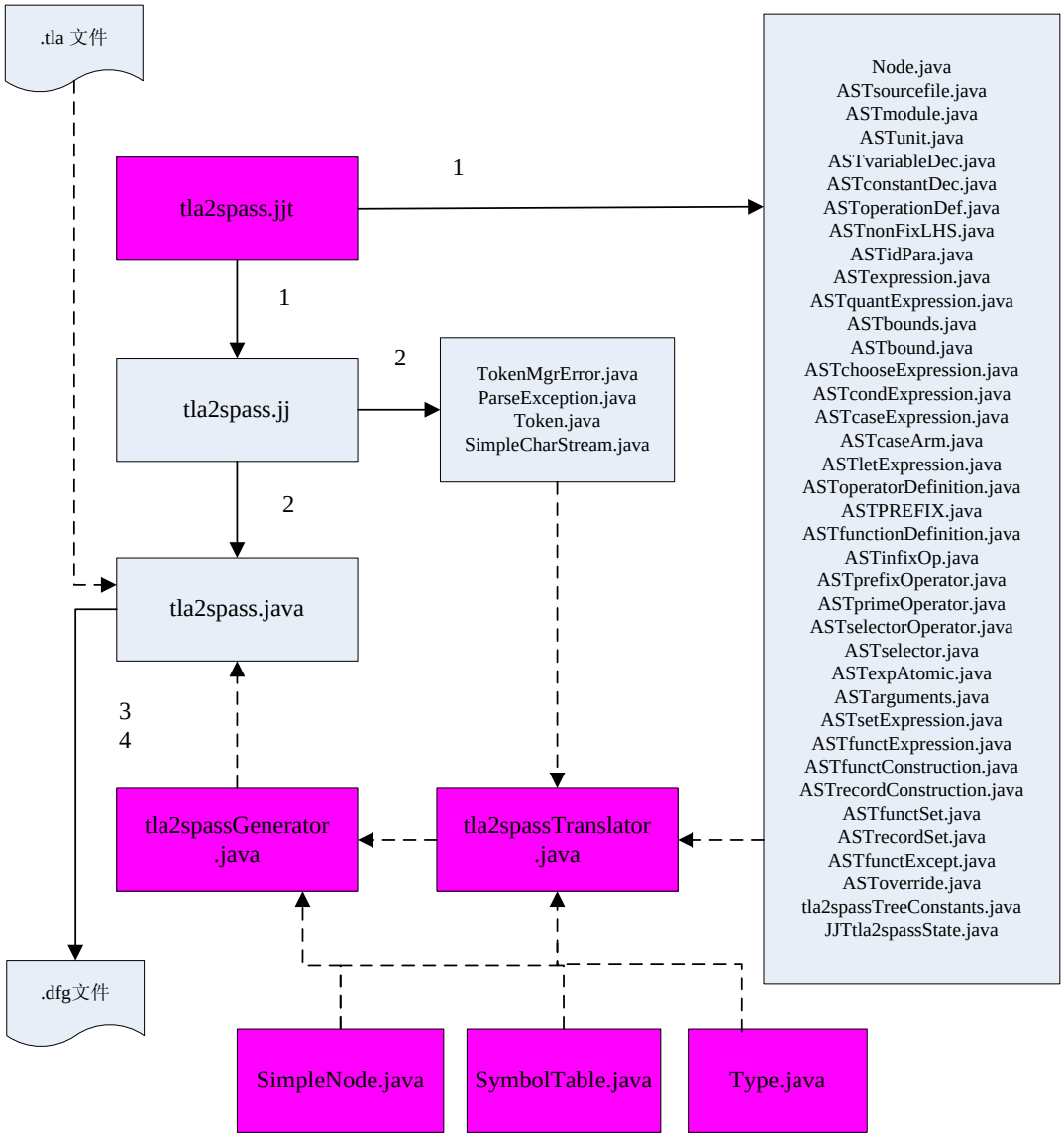

图 5.6　subTLA到SPASS转换的JavaCC实现涉及的文件

$tla2spass.java$文件是模型转换入口和出口，它以一个待转换.tla文件作为输入，生成对应的.dfg文件作为输出。图中的实线箭头表示文件的生成关系，箭头指向生成的文件；虚线箭头表示文件使用关系，箭头方向表示文件使用方向。图

中深色方框中的文件是我们为完成模型转换所编写的核心代码文件，包括6个文件：$tla2spass.jjt$、$tla2spassGenerator.java$ 、$tla2spassTranslator.java$、$Type.java$、$SimpleNode.java$以及$SymbolTable.java$。$tla2spass.jjt$的扩展名是jjtree的简称，这是java抽象语法树定义的文件类型。该文件包含了subTLA的词法和语法信息。此外，我们在语法分析过程中进行类型推演，这通过在$tla2spass.jjt$嵌入对应的java代码实现。得到的类型信息保存在$SymbolTable.java$中，供转换时使用。

图中浅色方框中的文件皆为javaCC工具生成的文件。$tla2spass.jj$文件是执行预处理命令$jjTree\ tla2spass.jjt$所得，同时生成一组以AST开头的抽象语法树数据结构定义文件。$tla2spass.java$是执行命令$javacc\ tla2spass.jj$所得，同时生成语法分析对应的辅助文件，这些辅助文件将用于实现完整的文件解析。$SimpleNode.java$文件表达了我们定义的抽象语法树，它继承自JavaCC生成的$Node$类，并添加了额外的属性和方法，以存储转换过程中需要的节点信息以及节点之间的关联信息。

$tla2spassTranslate.java$文件实现基本的转换过程，它通过查找各种表达式在语法树中的对应类型定位表达式，并根据表达式的转换规则，基于语法树节点信息生成对应的SPASS文本，存储于文件$SimpleTable.java$中。

$tla2spassGenerator.java$利用$tla2spassTranslate.java$翻译的SPASS文本信息，生成表示最终SPASS模型的.dfg文件。$Type.java$ 文件是类型推演中用到的类型数据结构（表示函数或谓词），用于类型推演过程。

转换流程为：

- 待处理文件（如$control.tla$）通过JavaCC生成的$tla2spass.java$进入转换流程，被javaCC解析为一个树形结构，同时保存对应变量的类型，并完成类型推演工作；
- 程序返还一个根节点，$tla2spass.java$程序调用 $tla2spassGenerator$类的$start$方法，并将之前生成的根节点作为参数。结果为SPASS输入格式的文件，被保存到与被翻译文件同名的.dfg文件中（如$control.dfg$）。
- $tla2spassGenerator$类除规范输出格式外，需要调用$tla2spassTranslator$类的$start$方法，将解析树的节点翻译成SPASS文本。
- $tla2spassTranslator$类的$start$方法完成具体的翻译工作。
- $tla2spassGenerator.java$ 和 $tlas2spassTranslator.java$ 两个文件都需用到

SimpleNode 类、*Type* 类和 *SimpleTable* 类来保存信息。

模型转换的命令执行过程为：

1. *jjTree tla2spass.jjt* 执行JJTree预处理，生成*tla2spass.jj*文件以及抽象语法树对应的一组辅助.java文件；
2. *javacc tla2spass.jj* 执行javaCC工具，生成*tla2spass.java*文件以及语法分析过程的辅助.java文件；
3. *javac * .java* 编译所有的java文件，得到可执行的.class文件；
4. *java tla2spass* ***.tla 执行 *tla2spass.class*，将***.tla文件转换为对应的***.dfg文件。

5.5 本章小结

根据前几章的研究成果，本章提出一种EDOLA语言的体系结构，并基于此体系结构构造了PLC领域建模验证语言EDOLA-PLC。为便于语言应用，开发了EDOLA-PLC 的工具原型，包括语言的编辑器和编译器两部分。编辑器提供了基本的用户操作界面；编译器实现了语言的语法、语义检查以及基于转换的自动验证功能。

第 6 章 EDOLA-PLC语言的应用实例

本章介绍基于EDOLA-PLC语言及其工具对一个典型案例：码头消防PLC控制系统进行建模和验证的实践。在建模方面，EDOLA-PLC可方便描述码头消防案例中的参数、输入输出变量、系统动作、实时约束以及待验证属性；PLC的循环扫描周期运行模式通过EDOLA-PLC语言的设计得到保证；环境模型无须用户编写，EDOLA-PLC编译器可自动生成环境模型并实现其与系统行为子模型的组合。在验证方面，码头消防案例的 8 条正确性属性通过EDOLA-PLC的编译器，分别基于模型检测和自动定理证明方法得到了验证。实验结果证实了编译过程中引入的面向验证需求抽象策略的有效性。与已有工作的比较进一步说明了EDOLA-PLC语言的特点及其与已有语言的关系和优势。

6.1 引言

本章介绍EDOLA-PLC语言及其工具的一个应用实例：码头消防PLC控制系统的建模和验证，以通过此应用案例，进一步解释和检验EDOLA领域知识层、公共模块层和验证支持层的设计以及研究成果。

本章的案例分析通过建模、验证和优化策略三个方面开展，以检验EDOLA-PLC的多方面特点。在建模方面，码头消防案例使用了EDOLA-PLC语言的常量声明、输入输出变量声明、系统动作定义、实时约束、待验证属性等多种语法成分，以检验EDOLA-PLC语言描述实际问题时的可用性和易用性；在验证方面，对于码头消防实例EDOLA-PLC模型，我们分别进行了基于模型检测和基于自动定理证明两种方法验证正确性属性的实验，以检验EDOLA-PLC的自动验证性。最后，本章还设计了对比试验，检验编译过程的面向需求抽象策略的有效性。

在实践检验的基础上，本章也从理论层面对EDOLA-PLC进行了定性分析。鉴于EDOLA-PLC的设计参考了多种类型的语言，其设计目标是实现领域描述易用性、复用性和自动验证性的综合优势，我们从现有形式语言、建模语言和传统DSL语言中选取了典型实例，与EDOLA进行了比较。最后，我们也将EDOLA的整体设计框架与现有的一些研究框架进行了对比分析。

本章的其余部分组织如下：第 6.2 节介绍码头消防PLC控制系统实例；第 6.3 节介绍基于EDOLA-PLC的码头消防PLC控制系统建模；第 6.4 节介绍基于转换的验证实践和实验结果；第 6.5 节介绍语言的相关比较；第 6.6 节是本章的小结。

6.2　码头消防PLC控制系统

本章介绍的码头消防PLC控制系统案例，是对“中化格力南迳湾公共码头消防控制系统”实际应用的抽象和简化。在此案例中，用户在控制台上输入控制命令，对各消防设备进行控制，并根据控制台上的状态显示，确定下一步动作；PLC控制软件在用户控制下操作消防设备并在操作台上显示当前操作状态。码头消防案例的物理设备配置如图 6.1 所示。码头有南北两个泊位（$berth1$ 和

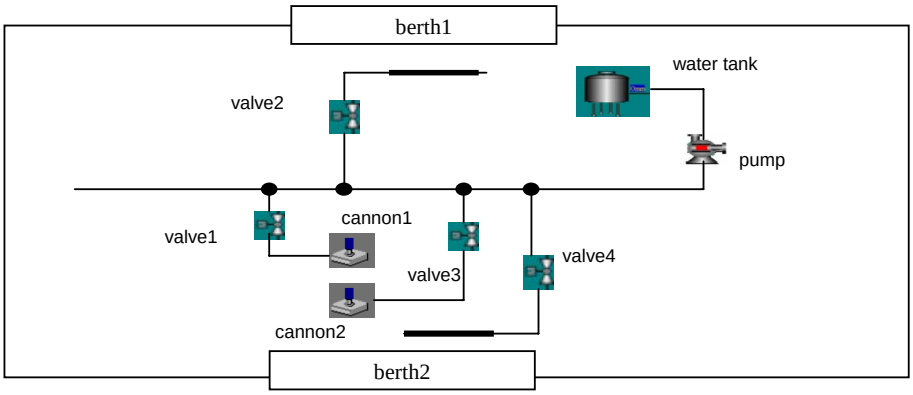

图 6.1　码头消防控制系统的物理连线示意图

$berth2$）来停泊船只，并配有两个水炮（$cannon1$ 和 $cannon2$）作为灭火装置。水泵（$pump$）用来从水罐（$watertank$）抽水，供应给水炮。各设备的连接线路配有管道阀门。例如，当阀门 $valve1$ 和 $valve2$ 都开启时，消防炮 $cannon1$ 可用于泊位$berth1$的消防活动。

控制面板主要由按钮、指示灯、操作杆、蜂鸣器等组成，与用户进行交互。用户通过按下按钮来发出命令；当前系统状态由面板上的系统状态指示灯标明；操作杆用来控制消防炮的移动方向，而蜂鸣器则用于异常报警。控制面板的示意图见图 6.2 。 图 6.2 的控制面板包含控制电源开关的一个自锁按钮和六个非

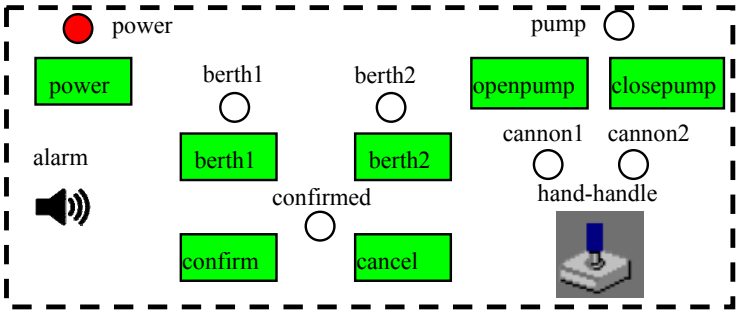

图 6.2　控制面板示意图

自锁按钮。六个非自锁按钮分别用于发出命令以开启水泵、关闭水泵、选择消防工况（$berth1$ 着火或$berth2$着火）、确认或取消当前设置。操作杆控制正在使用的水炮上、下、左、右移动。面板上的七个指示灯标明系统所在的不同状态，例如指示灯 $pump$ 亮时说明水泵已开启，灯灭时表明水泵已关闭。启用哪个消防炮，由用户对泊位的选择以及消防设备的物理配置共同决定。指示灯$cannon1$ 和 $cannon2$ 则表明当前正在使用的消防炮是哪一个。左侧的蜂鸣器（$alarm$）当水泵非正常开启时，进行报警。

码头失火时，用户应按照如下顺序在控制面板操作消防设备，进行灭火活动：(1) 开启电源（按下$power$ 按钮）；(2) 启动水泵（按下 $openpump$ 按钮）；(3) 选择要进行灭火的泊位 $berth1$ 或 $berth2$；(4) 确认选择（按下 $confirm$ 按钮）；(5) 移动操作杆以改变消防炮的方向进行灭火。

码头消防结束后，用户应执行如下系列动作以结束控制：(6) 关闭水泵（按下 $closepump$ 按钮）；(7) 结束当前消防工况（按下取消按钮 $cancel$ ）。最后，用户可继续其它的灭火行动（返回到步骤(2)）或抬起自锁按钮 $power$，关闭此消防控制系统。

实际的码头控制系统中还包含各种报警信息。本章考虑的简化后码头消防实例仅包含水泵开启的报警信息，即在控制软件发出开启水泵命令后5秒钟，水泵还未开启（例如由于设备故障），则控制面板上的蜂鸣器持续响3秒钟，并恢复控制系统到初始状态。

PLC控制软件检测用户的命令以及设备的状态，进行相应的计算以控制设备或显示系统信息。在码头消防案例中，PLC的主要任务是控制水泵的开启和关闭、阀门的开关、水炮的移动方向、蜂鸣器的开关以及控制面板上各状态指示灯

的点亮或熄灭。

6.3 码头消防控制系统的EDOLA-PLC模型

根据EDOLA-PLC的语法定义，本节从常量声明、变量声明、动作定义、约束定义和属性定义五部分，分别介绍码头消防PLC控制系统的EDOLA-PLC建模过程。

6.3.1 参数化描述

同实际系统一样，建模时也要考虑模型的可扩展性。例如，我们可将码头消防案例的EDOLA-PLC模型固定为两个泊位和现有的物理布线。然而，一般希望建立的模型能具有一定的灵活性和参数化特征，以便适应类似但不同的具体配置。哪些系统成分应被建模为可配置或可扩展的，将取决于系统设计者的决策，与具体问题密切相关。

对于此码头消防案例，我们从用户和设计者提供的信息得知，码头消防的控制流程通常不变，但根据资金情况，将来可能增加新的消防设备和消防工况；控制系统的设计应尽量对此提供支持。更进一步，当添加新设备或调整工况时，物理配置需进行相应的修改。因此，在建模时，我们将系统中的消防工况数目、消防炮数目、消防工况与消防炮的对应关系、阀门个数以及阀门与消防炮间的物理连接等设置为可配置的参数。参数声明在 EDOLA-PLC中通过**CONSTANT** 语句实现，表示为

$$\begin{aligned}\textbf{CONSTANT}\quad & FireCase, Cannon, Valve,\\ & CannonInCase(\_), BelongTo(\_)\end{aligned}$$

其中$FireCase$，$Cannon$ 和 $Valve$分别表示消防工况集合、消防炮集合和阀门集合。我们使用了集合而不是简单的自然数编号，以便描述清晰，这也是 EDOLA-PLC从基语言TLA$^+$中继承而来的优点：抽象描述能力。 $CannonInCase(\_)$ 和 $BelongTo(_)$是常函数，$CannonInCase(c) = i$ 表示消防炮 c 可在消防工况 i 中使用，$BelongTo(v) = c$ 表示阀门 v 出现在消防炮 c 的物理连接上。在描述具体系统时，只需将模型的参数实例化为具体值即可。例如，对于第 6.2 节描述的码头

消防案例，参数配置为

$FireCase$ = {“berth1”, “berth2”}

$Cannon$ = {“cannon1”, “cannon2”}

$Valve$ = {“valve1”, “valve2”, “valve3”, “valve4”}

$CannonInCase$(“cannon1”) = “berth1”

$CannonInCase$(“cannon2”) = “berth2”

$BelongTo$(“valve1”) = “cannon1”

$BelongTo$(“valve2”) = “cannon1”

$BelongTo$(“valve3”) = “cannon2”

$BelongTo$(“valve4”) = “cannon2”

6.3.2 变量声明

变量声明包括输入变量、输出变量和系统变量三部分。对于码头消防实例，输入变量包括用户控制台输入命令和设备反馈信号。用户控制台输入变量包括所有的按钮变量和操作杆变量。设备反馈仅包含表示水泵状态的变量。输出变量包括指示灯开关、水炮方向控制命令、阀门开关命令、水泵启停以及蜂鸣器开关命令等。系统变量仅包含一个系统状态变量。一种直观表示是将每一按钮、指示灯、阀门都表达为一个BOOLEAN 类型的变量，那么在码头消防实例模型中就至少需要14个变量显式表示7个按钮和 7个指示灯，而阀门和消防炮的个数是参数，所以这部分的BOOLEAN 变量个数未知。随着系统规模的增长，这种表示将会导致系统变量个数急剧增多，使得模型难以管理。为得到简洁可读的模型，我们采用数组来分别表示按钮、指示灯和阀门等。数组在EDOLA-PLC中通过函数来表示。完整的码头消防实例的变量声明为

INPUTVAR

$u\_button \in [Button \rightarrow \text{BOOLEAN}]$

$u\_handle \in Direction$

$realPump \in \text{BOOLEAN}$

OUTPUTVAR

$s\_buttonLight \in [ButtonLight \rightarrow \text{BOOLEAN}]$

$s\_cannonLight = [Cannon \rightarrow \text{BOOLEAN}]$

$s\_handle = [Cannon \rightarrow Direction]$

$valve = [Valve \rightarrow \text{BOOLEAN}]$

$pumpCtl \in \text{BOOLEAN}$

$alarm \in \text{BOOLEAN}$

SYSTEMVAR

$s\_sysState \in StateSet$

其中输入变量有3个：u_button是用户按钮数组变量，u_handle是用户操作杆变量，$realPump$是水泵状态变量；输出变量有6个：$s_buttonLight$是按钮指示灯开关数组变量，$s_cannonLight$是消防炮指示灯开关数组变量，s_handle是操作杆控制命令数组变量，$valve$是阀门开关数组变量，$pumpCtl$是水泵启停控制变量，$alarm$是蜂鸣器开关控制变量；系统变量$s_sysState$表示系统状态。$Cannon$和 $Valve$ 是常量标识符。$StateSet$ 表示系统状态变量的取值集合；$Direction$ 表示消防炮的取值方向集合；$Button$ 表示按钮集合，包括电源自锁按钮和所有非自锁按钮；指示灯区分为按钮指示灯和消防炮指示灯两部分，$ButtonLight$ 代表按钮对应的指示灯集合。这些集合定义在EDOLA-PLC的辅助定义部分给出，如下所示。

$StateSet$ == {“init”, “power”, “openpumpSent”, “pumpopened”, “selected”, “cannonOnUse”, “closepumpSent”, “pumpclosed”}

$Direction$ == {“up”, “down”, “left”, “right”, “none”}

$SelfLockedButton$ == {“power”}

$UnlockedButton$ == {“openpump”, “closepump”, “confirm”, “cancel”} $\cup$ $FireCase$

$Button$ == $SelfLockedButton \cup UnlockedButton$

$ButtonLight$ == {“power”, “pump”, “confirmed”} $\cup$ $FireCase$

6.3.3 动作定义

动作定义包括初始状态、动作序列和规范说明三部分。初始状态定义通过设置输入变量、输出变量和系统变量的初始值完成。在码头消防实例的初始状态下，对于输入变量，所有按钮都未被按下，假定用户操作杆处于中间位置“none”，设备变量$realPump$ 初始值设置为假，表示未启动；对于输出变量，所有指示灯初始都处于熄灭状态，开启水泵命令尚未发送，消防炮方向设置在中间位置；系统状态变量初始设置为“init”。模型的初始状态基于EDOLA-PLC语言表示为

INIT

$$
\begin{aligned}
Init \triangleq\ & \wedge\ u\_button = [i \in Button \mapsto \text{FALSE}] \\
& \wedge\ u\_handle = \text{“none”} \\
& \wedge\ realPump = \text{FALSE} \\
& \wedge\ s\_sysState = \text{“init”} \\
& \wedge\ s\_buttonLight = [i \in ButtonLight \mapsto \text{FALSE}] \\
& \wedge\ s\_cannonLight = [i \in Cannon \mapsto \text{FALSE}] \\
& \wedge\ s\_handle = [i \in Cannon \mapsto \text{“none”}] \\
& \wedge\ valve = [i \in Valve \mapsto \text{FALSE}] \\
& \wedge\ pumpCtl = \text{FALSE} \\
& \wedge\ alarm = \text{FALSE}
\end{aligned}
$$

根据自然语言描述，动作序列部分包含12个可能的动作，分别表示开启电源、开启水泵、接收水泵开启信号、报警、报警后状态重置、选择消防工况（泊位）、确认选择、接收消防炮方向信号、关闭水泵、接收水泵关闭信号、结束消防工况处理和关闭电源。基于EDOLA-PLC的动作定义形式为

ACTION

$$
\begin{aligned}
& PowerUp == \ldots \\
& OpenPump == \ldots \\
& PumpLightOn == \ldots \\
& Alarm == \ldots \\
& AlarmReset == \ldots
\end{aligned}
$$

$\exists i \in FireCase : SelectCase(i) == \ldots$

$Confirm == \ldots$

$ResponseHandle == \ldots$

$ClosePump == \ldots$

$PumpLightOff == \ldots$

$Cancel == \ldots$

$PowerDown == \ldots$

为描述简洁，本节仅介绍开启水泵动作 $OpenPump$ 的定义，其它动作定义与之类似。开启水泵动作的使能条件是：电源已开启、用户按下了开启水泵按钮*openpump*且没有任何其它按钮被按下。此动作基于EDOLA-PLC描述为

$OpenPump ==$

$\wedge\ s\_sysState$ = “power”

$\wedge\ u\_button$[“power”]

$\wedge\ u\_button$[“openpump”]

$\wedge \forall i \in UnlockedButton : i \neq$ “openpump” $\Rightarrow \neg u\_button[i]$

$\wedge\ pumpCtl'$

$\wedge\ s\_sysState'$ = “openpumpSent”

其中，由合取符连接的前4行，表达了动作 $OpenPump$ 的使能条件；第5至6行表达了执行此动作所带来的效果。具体来讲，当系统处于“power”状态（第1行）、自锁按钮 $power$ 已按下（第 2行）、用户按下了 $openpump$ 按钮（第3行）且此时无任何其它非自锁按钮按下（第4行）时，动作 $OpenPump$ 被使能。动作执行时，将发送一个开启水泵命令（第5行），将系统状态设置为“openpumpSent”（第6行），并保持所有其它变量不变。保持所有其它变量不变这一层含义隐含在EDOLA-PLC语言的语义中，而TLA$^+$则需显式通过UNCHANGED 操作符描述所有保持不变的变量。从这一点看，EDOLA-PLC的表达更简洁。

规范说明部分指定码头消防实例EDOLA-PLC模型的名字，由语句 **SPEC** $Control$ 给出。

6.3.4 约束定义

约束定义包括环境约束和实时约束两部分。码头消防控制实例使用了全环境模式，因而在EDOLA-PLC中只需使用关键字 **ENV TOTAL** 描述，EDOLA-PLC工具将自动生成环境模型并实现与系统动作子模型的组合。实时约束包含与水泵开启动作相关的报警信息，它涉及到系统的开启水泵动作、报警动作和报警后重置三个动作，在EDOLA-PLC的动作定义分别表示为*OpenPump*、*Alarm*和*AlarmReset*。消防实例中的水泵开启5秒超时约束可通过EDOLA-PLC的 **TIMEOUT** 操作符描述，表达为 **TIMEOUT**($OpenPump, Alarm, 5, t1, t2$)，其中$t1$和$t2$是两个动作所对应的时钟名；蜂鸣器持续响3秒的实时约束可通过 **DELAY** 操作符表示，描述为 **DELAY**($AlarmReset, 3, t3$)，其中$t3$是用户自定义的时钟名。

6.3.5 属性表达

为验证码头消防实例的设计是否正确，一般先通过排错性质对模型进行基本的调试，调试通过后再对正确性属性进行验证。码头消防实例的EDOLA-PLC模型中可设置12条排错属性，以分别检查12个系统动作$PowerUp$、$OpenPump$、$PumpLightOn$、$Alarm$、$AlarmReset$、$\exists i \in FireCase : SelectCase(i)$、$Confirm$、$ResponseHandle$、$ClosePump$、$PumpLightOff$、$Cancel$和 $PowerDown$是否可执行。它们将逐条在EDOLA-PLC的属性定义部分表示。例如，消防工况选择动作的排错检查属性表示为：

DEBUG $CheckSelectCase : \exists i \in FireCase : SelectCase(i)$

正确性属性基于EDOLA-PLC语言提供的各种操作符进行定义。本节列举出码头消防实例的8条属性以验证模型的正确性，它们在EDOLA-PLC的 **TOPROOF** 属性部分表达。这8条属性的自然语言描述以及基于EDOLA-PLC的表示分别介绍如下。

1. 关闭水泵正确响应属性 码头消防实例中，若系统所处的不是水泵应被关闭的状态（$s\_sysState$ = “cannonOnUse”，那么即使用户输入关闭水泵命令（u_button[“closepump”]），关闭水泵动作（$ClosePump$）也不应被使能。我们

将此属性命名为 $ClosePumpNotRespond$，它可基于**RESPOND**关键字进行定义

$ClosePumpNotRespond$：

RESPOND($ClosePump$, u_button[“closepump”], $s\_sysState$ = “cannonOnUse”)

2. 消防工况正确响应属性　若系统未处于对应的状态，消防工况的选择按钮将不被响应。此属性命名为$SelectCaseNotRespond$，基于 **RESPOND** 关键字表示为

$$\begin{aligned} &SelectCaseNotRespond: \\ &\quad \textbf{RESPOND}(\ SelectCase(i), \forall i \in FireCase : u\_button[i], \\ &\qquad\qquad s\_sysState \in \{\text{“pumpopened”}, \text{“selected”}\}\ ) \end{aligned}$$

3. 消防工况选择的唯一性　受物理资源限制，在码头消防实例中，要求任一时刻最多只能有一个消防工况被选择。此性质属于竞争属性，可基于关键字**COMPETE**表达为

$$\begin{aligned} &CaseSelectOnlyOne : \forall i \in Firecase, j \in Firecase : \\ &\quad \textbf{COMPETE}(\ i \neq j, s\_buttonLight[i], s\_buttonLight[j]\ ) \end{aligned}$$

4. 消防炮使用的唯一性　码头消防实例中，同一时刻最多只有一个消防炮在用于消防灭火行动。此性质同样属于竞争属性，表示为

$$\begin{aligned} &CannonUsedOnlyByOne : \forall i \in Cannon, j \in Cannon : \\ &\quad \textbf{COMPETE}(\ i \neq j, s\_cannonLight[i], s\_cannonLight[j]\ ) \end{aligned}$$

5. 阀门开启的互斥性　根据消防炮的唯一性，进一步可验证用于不同消防炮的阀门之间的互斥性。即在任一时刻，若多个阀门都处于开启状态，则它们必归属于同一个消防炮的物理连接。这也是一条竞争属性，在EDOLA-PLC中表示为

$$\begin{aligned} &ValveMutex : \forall i \in Valve, j \in Valve : \\ &\quad \textbf{COMPETE}(\ BelongTo(i) \neq BelongTo(j), valve[i], valve[j]\ ) \end{aligned}$$

6. 水泵开启和电源开启的顺序性 第 6.2 节码头消防实例的非形式化描述指定了期望的动作响应顺序。例如，开启水泵和开启电源之间的顺序关系。具体来讲，只有在电源开启之后（从而电源灯亮），水泵开启动作才能执行。此属性可基于顺序属性操作符 **SEQUENCE** 表示为

$$OpenPumpAfterPower:$$
$$\textbf{SEQUENCE}(\ OpenPump,\ s\_buttonLight[\text{"}power\text{"}]\ )$$

7. 消防工况选择与水泵开启的顺序性 另一个顺序属性的例子是消防工况选择动作只能在水泵已开启的情况下才能被执行。与属性 6 类似，表达为

$$SelectAfterOpenPump : \forall i \in FireCase :$$
$$\textbf{SEQUENCE}(\ SelectCase(i),\ s\_buttonLight[\text{"}pump\text{"}]\ )$$

即消防工况i仅在水泵的指示灯亮（从而水泵已开启）之后才能被选择。

8. 电源关闭动作的优先性 码头消防实例中，电源按钮 $power$ 具有最高的优先级。因此，在任何系统状态下，只要用户在控制面板上抬起电源按钮，则系统电源关闭，不能再执行任何其它动作。此属性可通过EDOLA-PLC的优先属性关键字**PRIORITY**表达为

$$ClosePowerAlwaysRespond:$$
$$\textbf{PRIORITY}(\ PowerDown, s\_buttonLight[\text{"}power\text{"}] \wedge \neg u\_button[\text{"}power\text{"}]\ )$$

此属性的直观解释为，若用户在电源已开启的情况下，抬起了电源按钮，则只有 $PowerDown$ 动作被使能而所有其它的系统动作都是非使能的，从而电源关闭，系统停止动作。

6.4 码头消防EDOLA-PLC模型的自动验证

EDOLA的自动验证通过基于中间语言subTLA的转换实现。EDOLA-PLC工具中实现了EDOLA-PLC模型到subTLA模型、subTLA模型到模型检测工具TLC以及subTLA模型到自动定理证明工具SPASS的无缝链接，因此用户不

必关注中间生成的subTLA模型。然而，为进一步对基于中间语言的验证支持进行解释，本节首先介绍码头消防实例EDOLA-PLC模型编译后的subTLA表示，再分别介绍基于模型检测工具TLC和自动定理证明工具SPASS的自动验证结果。

6.4.1　EDOLA-PLC模型到subTLA模型的编译结果

根据第 5 章EDOLA-PLC的语法定义和第 2 章PLC领域操作符的语义解释，很容易得到EDOLA-PLC到subTLA的编译规则。码头消防实例EDOLA-PLC模型的模块开始、模块结束、模块扩展以及常量定义，可直接对应到subTLA中的表示，无须转换。EDOLA-PLC的变量声明包含类型信息，它被转换为subTLA中的变量声明

VARIABLES

$$u\_button, u\_handle, realPump,$$
$$s\_sysState, s\_buttonLight, s\_cannonLight, s\_handle, valve, pumpCtl, alarm,$$
$$aux$$

和一个类型不变式

$$\begin{aligned} TypeInvariant \triangleq\ & \wedge\ u\_button \in [Button \rightarrow \text{BOOLEAN}\ ] \\ & \wedge\ u\_handle \in Direction \\ & \wedge\ realPump \in \text{BOOLEAN} \\ & \wedge\ s\_buttonLight \in [ButtonLight \rightarrow \text{BOOLEAN}\ ] \\ & \wedge\ s\_cannonLight \in [Cannon \rightarrow \text{BOOLEAN}\ ] \\ & \wedge\ s\_handle \in [Cannon \rightarrow Direction] \\ & \wedge\ s\_sysState \in StateSet \\ & \wedge\ valve \in [Valve \rightarrow \text{BOOLEAN}\ ] \\ & \wedge\ pumpCtl \in \text{BOOLEAN} \\ & \wedge\ alarm \in \text{BOOLEAN} \\ & \wedge\ aux \in \{0, 1, 2\} \end{aligned}$$

其中，*aux*是编译过程中引入的辅助变量，用来构成subTLA中对PLC循环扫描模式的表示。

现在介绍动作定义部分的转换。EDOLA-PLC初始状态定义到subTLA的转换由原初始状态定义合取上辅助变量*aux*初始值的设置 $aux = 0$构成。由于subTLA中对系统不修改的变量需要显式表达，因此一个EDOLA-PLC动作的subTLA表示，由原动作定义合取上一条UNCHANGED 语句构成。UNCHANGED 语句中包含动作定义中未对其进行修改的变量，即未以带撇变量形式出现的系统变量和输出变量。系统动作定义也不修改任何输入变量的值，这部分信息将在subTLA的PLC循环扫描模式中实现。例如，$OpenPump$ 动作对应的subTLA动作表示为

$$
\begin{aligned}
&OpenPump \triangleq \\
&\land\ s\_sysState = \text{“power”} \\
&\land\ u\_button[\text{“power”}] \\
&\land\ u\_button[\text{“openpump”}] \\
&\land\ \forall i \in UnlockedButton : i \neq \text{“openpump”} \Rightarrow \neg u\_button[i] \\
&\land\ pumpCtl' \\
&\land\ s\_sysState' = \text{“openpumpSent”} \\
&\land\ \text{UNCHANGED}\ \langle s\_buttonLight, s\_cannonLight, s\_handle, valve, alarm\rangle
\end{aligned}
$$

规范说明部分在EDOLA-PLC中简单地表示为一个名字*Control*，它将根据PLC领域知识，转换为subTLA的规范说明定义以及一组相关的公式。此编译过程与环境约束、实时约束及属性定义都紧密相关。为使编译后的subTLA模型能被自动验证工具有效验证，编译过程引入了面向属性的抽象策略。码头消防实例EDOLA-PLC模型的属性定义中不包含实时属性，因此编译过程中将忽略由关键字 **TIME** 引出的实时约束部分的转换。从而，EDOLA-PLC的规范说明 **SPEC** $Control$ 转换为下面的subTLA规范说明和一组相关的公式

$$Control \triangleq Init \land \Box[Next]_{vars}$$

码头消防实例未指定公平性约束，因此subTLA范式中的 L 被忽略。$Init$ 为初始状态定义。码头消防实例采用了全环境约束，因此系统动作 $Next$ 被解释为包含

环境重置动作 *ClearEnvInput* 的版本（参见第 2.4.4.3 节的介绍）

$$
\begin{aligned}
Next \triangleq \lor & \land aux = 0 \\
& \land EnvInput \\
& \land \textsc{unchanged}\ SOV \\
& \land aux' = 1 \\
\lor & \land aux = 1 \\
& \land SystemAction \\
& \land \textsc{unchanged}\ IV \\
& \land aux' = 2 \\
\lor & \land aux = 2 \\
& \land ClearEnvInput \\
& \land \textsc{unchanged}\ SOV \\
& \land aux' = 0
\end{aligned}
$$

公式中的 SOV 是所有系统变量和输出变量的集合，IV是输入变量集，它们分别定义为

$$SOV \triangleq \langle s\_sysState, s\_buttonLight, s\_cannonLight, s\_handle, valve, pumpCtl, alarm \rangle$$

$$IV \triangleq \langle u\_handle, u\_button, realPump \rangle$$

$SystemAction$ 是PLC响应动作，表达为公式

$$
\begin{aligned}
SystemAction \triangleq \lor & \ React \\
\lor & \land \ \textsc{enabled}\ React \\
& \land \textsc{unchanged}\ SOV
\end{aligned}
$$

其中 $React$ 是**ACTION**定义部分所有动作的析取，表示为

$$
\begin{aligned}
React \triangleq \lor & \ PowerUp \\
\lor & \ OpenPump \\
\lor & \ PumpLightOn \\
\lor & \ Alarm
\end{aligned}
$$

$$\vee\ AlarmReset$$
$$\vee\ \exists i \in FireCase : SelectCase(i)$$
$$\vee\ Confirm$$
$$\vee\ ResponseHandle$$
$$\vee\ ClosePump$$
$$\vee\ PumpLightOff$$
$$\vee\ Cancel$$
$$\vee\ PowerDown$$

属性描述部分，根据第 2 章各属性操作符的TLA$^+$形式语义进行翻译。例如，第 6.3.5 节定义的属性 1 和属性 3 对应的subTLA表示，分别为

$$CloseePumpNotRespond \triangleq$$
$$\Box(u\_button[\text{“closepump”}] \wedge s\_sysState \neq \text{“cannonOnUse”}$$
$$\Rightarrow \neg(\textsc{enabled}\ ClosePump))$$

和

$$CaseSelectOnlyOne \triangleq$$
$$\Box(\forall i, j \in FireCase : s\_buttonLight[i] \wedge s\_buttonLight[j] \Rightarrow i = j)$$

6.4.2 基于TLC的模型检测结果

本节介绍基于模型检测工具TLC对码头消防EDOLA-PLC模型自动验证的实验结果。模型检测工具TLC不能直接处理带参模型，因此我们考虑的是码头消防实例所对应的具体模型，参数配置描述见第 6.3.1 节的介绍。下面的实验使用的是2008年4月发布的TLC版本；实验机器cpu型号为Intel® Core™ 2 duo, T8100 2.10GHz，内存为3GB。

对于排错属性，我们期待TLC返回结果为FALSE。针对码头消防实例，本章分别检测了表 6.1 中定义的 12 条排错属性，它们根据EDOLA-PLC的排错属性描述在编译过程中生成。对于这 12 条属性，TLC成功返回期望的结果 FALSE 并对检测的每一个动作，报告了一条能够使其发生的路径。

表 6.1 码头消防案例12条排错属性的subTLA表示

编号	排错属性定义
1	$CheckPowerUp \triangleq \Box[\neg PowerUp]_{vars}$
2	$CheckOpenPump \triangleq \Box[\neg OpenPump]_{vars}$
3	$CheckPumpLightOn \triangleq \Box[\neg PumpLightOn]_{vars}$
4	$CheckAlarm \triangleq \Box[\neg Alarm]_{vars}$
5	$CheckAlarmReset \triangleq \Box[\neg AlarmReset]_{vars}$
6	$CheckSelectCases \triangleq \Box[\neg(\exists i \in FireCase : SelectCase(i))]_{vars}$
7	$CheckConfirm \triangleq \Box[\neg Confirm]_{vars}$
8	$CheckResponseHandle \triangleq \Box[\neg ResponseHandle]_{vars}$
9	$CheckClosePump \triangleq \Box[\neg ClosePump]_{vars}$
10	$CheckPumpLightOff \triangleq \Box[\neg PumpLightOff]_{vars}$
11	$CheckCancel \triangleq \Box[\neg Cancel]_{vars}$
12	$CheckPowerDown \triangleq \Box[\neg PowerDown]_{vars}$

对于所有正确性属性，我们期望TLC返回结果为 TRUE 。第 6.3.5 节码头消防实例的 8 条正确性属性，全部成功通过了TLC的验证，返回结果为真。此过程生成了 114,037 个状态，其中有 66,713 个不同状态，验证时间为 21.4 秒（不包括subTLA 模型的生成时间）。

为检验EDOLA-PLC到subTLA的编译阶段引入面向验证需求的抽象策略的有效性，本章设计了引入抽象策略与无抽象策略情形的对比实验。在无抽象策略情况下，须对关键字 **TIME** 引出的实时约束进行编译。根据第 4 章介绍的方法，我们首先建立仅包含领域知识层操作符（即不包含实时特性）的码头消防实例subTLA模型 $DomainControl$：

$$DomainControl \triangleq Init \wedge \Box[Next]_{vars}$$

其中，$Init$ 和 $Next$ 以及 $vars$ 的定义与前面的介绍完全相同。从而，EDOLA-PLC的规范说明 **SPEC** $Control$ 被编译为一个新的subTLA模块RTdock，定义如图 6.3 所示。此subTLA模型的规范说明由 $Control$ 定义，它由 $DomainControl$ 中的功能描述与实时约束描述组合而成。在根据公共模块层定义的时间操作符解释EDOLA-PLC的实时约束时，每个EDOLA-PLC描述的系统动作Act，被自动解释为一个subTLA动作F_{Act}，其形式与循环扫描模式中的系统响应部分相同，只是动作由$SystemAction$修改为一个具体的动作Act。也就是说，当考虑了实时

MODULE $RTdock$

EXTENDS $DomainControl$, $RealTimeNew$, $Sequences$

VARIABLE $t1$, $t2$, $t3$

$$
\begin{aligned}
BigInit \triangleq\ & \land Init \\
& \land now = 0 \\
& \land t1 = 0 \\
& \land t2 = 0 \\
& \land t3 = 0 \\
F\_PumpLightOn \triangleq\ & \land aux = 1 \\
& \land PumpLightOn \\
& \land aux' = 2 \\
& \land \text{UNCHANGED } IV \\
F\_Alarm \triangleq\ & \land aux = 1 \\
& \land Alarm \\
& \land aux' = 2 \\
& \land \text{UNCHANGED } IV \\
F\_AlarmReset \triangleq\ & \land aux = 1 \\
& \land Alarm \\
& \land aux' = 2 \\
& \land \text{UNCHANGED } IV \\
BigNext \triangleq\ & \land (Next \lor \text{ UNCHANGED } vars) \\
& \land (NowNext(vars) \lor now' = now) \\
& \land Timeout(F\_PumpLightOn,\ F\_Alarm,\ 5,\ t1,\ t2,\ vars) \\
& \land Delay(F\_AlarmReset,\ 3,\ t3,\ vars) \\
RTvars \triangleq\ & \langle t1,\ t2,\ t3,\ now \rangle \circ vars \\
RTL \triangleq\ & RTFairness(vars) \\
Control \triangleq\ & BigInit \land \Box[BigNext]_{RTvars} \land RTL
\end{aligned}
$$

图 6.3 无抽象策略的码头消防实例subTLA模型

约束时，用户基于EDOLA-PLC描述的一个动作Act等价于基于subTLA描述的一个动作F_{Act}。这是由于PLC的循环扫描周期以及系统不做修改的那些变量被隐含在EDOLA-PLC的语义上，而这部分在subTLA中则需显式表达。

下面尝试在此模型上验证码头消防实例的 8 条正确性属性。为能够应用模型检测工具TLC，首先需将时间描述由连续模型（由 Real表示）修改为一个有界的离散模型，这里通过将时间变量的取值范围限定为 0..10 实现，即设定一个整数区间，最大值为10。TLC同样验证了所有 8 条属性成立，但却花费了 359.2 秒的时间。这个过程中TLC生成了 5,693,250 个状态，其中有 763,918 个不同状态。此对比结果验证了采用抽象策略的有效性。实验结果同时也说明了TLC作为一个普通的模型检测工具，在验证时间模型时的低效。Lamport[125]指出，通过对实数值时钟的整数离散化（Integer Discretization）以及合适的抽象策略，TLC能够达到与UPPAAL相似的效率实现对时间模型的验证。目前EDOLA-PLC到subTLA的编译过程仅实现了时间约束和功能描述分离的抽象策略，以确保在包含时间约束的EDOLA-PLC模型上可以有效验证功能属性。对于包含时间特征的性质验证，将来可在编译过程中加入新的抽象策略来优化编译过程，如考虑从绝对时间到相对时间表示的抽象策略。

6.4.3 基于自动定理证明器SPASS的验证结果

自动定理证明工具可对带参数的EDOLA-PLC模型直接验证。在实际应用中，模型检测和定理证明互为补充。当验证参数化（无穷）系统时，可先基于模型检测技术验证系统的有限实例，以便发现错误或增强系统正确的信心，在此基础上再应用定理证明方法直接对参数化模型进行验证[126]。由于自动定理证明方法通常用于证明而不是排错，因此本章仅考虑 8 条正确性属性的验证。EDOLA-PLC的自动定理证明方法参见第 4 章的介绍。实验的验证工具为带等式的一阶定理证明工具SPASS，机器配置与第 6.4.2 节完全相同，工具版本为SPASS 3.0。8 条正确性属性基于EDOLA-PLC工具逐条进行验证，结果见表 6.2 。

在表 6.2中，$Inv \triangleq (s\_sysState = \text{“power”}) \Rightarrow s\_buttonLight[\text{“power”}]$。由表 6.2 可知，这 8 条正确性属性均可在较短的时间内基于SPASS工具自动证明。值得注意的是，基于第 4 章介绍的方法，可直接在subTLA模型上对归纳不变式进行证明。而对于非归纳不变式，则需手工寻找一个更强的归纳不变

表 6.2 基于SPASS的8条正确性属性证明结果

待验证属性	证明结果	SPASS运行时间
1. ClosePumpNotRespond	成立	12.4秒
2. SelectCaseNotRepsond	成立	10.6秒
3. CaseSelectOnlyOne	成立	1分33秒
4. CannonUsedOnlyByOne	成立	39.2秒
5. ValveMutex	成立	18分33秒
6. OpenPumpAfterPower	成立	1分37秒
$OpenPump \Rightarrow s\_buttonLight$[“power”]	不成立	17.2秒
$\Box Inv$	成立	1分26秒
$OpenPump \wedge Inv \Rightarrow s\_buttonLight$[“power”]	成立	10.3秒
7. SelectAfterOpenPump	成立	1分49秒
8. ClosePowerAlwaysRespond	成立	16.2秒

式来完成证明。以属性 $OpenPumpAfterPower$ 为例，当试图证明$OpenPump \Rightarrow s_buttonLight$[“power”] 时，SPASS的证明结果显示此公式不成立。经过分析，我们发现原因在于该公式的前提 $OpenPump$ 并未指定变量 $s\_sysState$ 和 $s\_buttonLight$ 之间的关系。因此，这里首先验证一个表明这两个变量之间关系的不变式 Inv，再通过验证属性$OpenPump \wedge Inv \Rightarrow s_buttonLight$[“power”] 来证明 $OpenPumpAfterPower$ 。这两个证明步骤分别在数十秒之内通过了SPASS的证明，从而证明了$OpenPumpAfterPower$ 成立。$SelectAfterOpenPump$ 的证明与之类似。$CannonUsedOnlyByOne$的证明使用了已证明的属性$CaseSelectOnlyOne$。

6.5 相关比较

前几节介绍了应用EDOLA-PLC语言对码头消防实例进行建模和验证的工作。本节给出这部分实践工作、EDOLA-PLC语言以及EDOLA设计方法与相关工作的比较。

6.5.1 与已有PLC建模验证工作的比较

PLC控制系统的描述和验证近年来受到广泛关注。很多研究者关注于PLC程序的建模和验证。Mader等[127]对PLC的模型进行了分类。 Heiner等[128]给出了指令集语言IL的Petri网语义； Canet等[129]通过将IL转换为符号模型检测工

具SMV的输入来对PLC程序进行验证；Willems等[130]、周旻等[80]研究了从IL语言到时间自动机的转换方法，并利用工具UPPALL对程序进行验证。由于他们的工作考虑的是最底层代码，因此得到的模型庞大，难以处理较大的实例。王瑞等[78]以舞台控制系统为例，建立了PLC系统的时间自动机模型，并验证了多种系统属性。他们的工作考虑较高的抽象层，直接使用时间自动机的同步操作符来描述环境和PLC的交互，未体现PLC的特征，因此模型与代码实现差距较大。基于EDOLA-PLC语言对PLC系统建模和验证的工作，模型抽象层次适中，既描述了PLC的循环周期运行模式，又避免了底层代码描述的繁琐。

6.5.2 EDOLA-PLC语言的相关比较

与形式语言的比较 通常认为基于状态变迁语义的形式语言如时间自动机和Petri网，较适合建模如PLC这类反应式系统的行为。另一方面，模型检测工具如NuSMV和 Spin等在PLC验证实践中得到了较多的应用，从而使得模型检测工具的输入语言如NuSMV的输入语言（在下面的描述中称其为iNuSMV）以及SPIN的输入语言Promela等常被用作PLC控制系统的建模语言。与它们不同的是，EDOLA-PLC的基语言是TLA$^+$，从而继承了TLA$^+$的很多优点。下面以iNuSMV为例，介绍EDOLA-PLC与它的定性比较，其它面向模型检测的形式语言与之有类似的结论。

首先，EDOLA-PLC的表达能力更强，它支持对无限情形的表达，如实数；而ilNuSMV为了模型检测的需要，只支持对有限情形的表达。在码头消防案例中，EDOLA-PLC中扩展了TLA$^+$中定义的Real模块来表示实数。

其次，EDOLA-PLC支持可配置的模型描述。EDOLA-PLC中的参数不仅包括常量，还包括常函数。码头消防实例中，当添加了新的消防工况或调整了物理配置时，仅参数实例化部分需要更新，整个模型无需修改。 EDOLA-PLC中的“类型”概念本质上是通过集合表达的，因此更加灵活。如码头消防实例中，EDOLA-PLC用$FireCase \triangleq$ {“berth1”, “berth2”}来实例化两个消防工况；也可使用 $FireCase \triangleq \{1, 2\}$来实例化，模型方面不需要任何改动。ilNuSMV 不支持类似的机制。

第三，基于EDOLA-PLC的模型更抽象和简洁。EDOLA-PLC继承了TLA$^+$中的抽象表达方法，用函数来表示数组，可直接对函数赋值，使得模型简洁。如

码头消防实例中的一个描述：当选择了消防工况 i 之后，其指示灯亮，所有其它工况的指示灯灭，而消防工况之外的所有其它指示灯保持不变。这个情形在EDOLA-PLC中可简洁地描述为

$$s\_buttonLight' = [j \in ButtonLight \mapsto \text{IF } j \notin FireCase \text{ THEN } s\_buttonLight[j] \text{ ELSE IF } j = i \text{ THEN TRUE ELSE FALSE}]$$

当尝试使用ilNuSMV对它进行描述时，会发现所得描述很繁冗。

最后，由于EDOLA-PLC基于TLA$^+$的逻辑框架进行设计，因此对EDOLA-PLC模型，既可以给出其模型检测方法又可以给出其定理证明方法，而ilNuSMV不具有这方面的优势。

与基语言TLA$^+$的比较 EDOLA-PLC是面向PLC领域的建模验证语言，因此，TLA$^+$更通用而EDOLA-PLC更擅长表达PLC领域知识。从EDOLA-PLC的抽象语法和码头消防实例可看出，EDOLA-PLC语言中直接提供了对循环扫描周期、环境约束、实时约束等的描述，将领域的语义固化在语言的语法成分中，表达领域知识时简洁易用。另外，TLA$^+$中要求显式表达保持不变的变量（体现在UNCHANGED 语句），而一般情况下，PLC的一个控制动作，只修改系统中的小部分变量，此时基于TLA$^+$语言的动作描述却需要包含一个很长的UNCHANGED 部分，使得模型编写繁冗，且影响模型的清晰性。EDOLA-PLC中隐式表达未修改的变量，从而省去了显式撰写这部分的工作量。另一方面，由于EDOLA-PLC基于TLA$^+$的一个子集subTLA描述其形式语义，因此在表达能力上TLA$^+$更强。EDOLA-PLC牺牲一部分表达能力，是为了自动验证的需要。理论上讲，EDOLA-PLC模型都可基于模型检测或自动定理证明进行自动验证，而对完整的TLA$^+$ 语言则不可能完全自动验证。

与其它DSL的比较 EDOLA-PLC与其它DSL的不同，主要体现在EDOLA-PLC的验证特征上。大部分DSL的设计目标都是可执行，而不考虑形式验证。考虑形式验证的DSL，一般通过转换的方法基于自动验证工具进行验证。其中，Hanna等[131,132]提出的面向传感器网络安全协议代码的验证框架Slede，与EDOLA-PLC的语言设计比较相似。Slede框架的一个特点是自动生成入侵模

型，实现入侵模型和协议代码模型的组合，从而避免了手工建立入侵模型所耗费的精力和引入的错误。这一点与EDOLA-PLC语言中通过**ENV TOTAL**关键字所表达的全环境相似：EDOLA-PLC在实现时会自动生成环境模型，并通过 PLC循环扫描模式实现环境模型和PLC系统响应模型的组合，不同之处在于EDOLA-PLC 提供了自动生成和用户自定义两种环境模型定义方式。Slede和EDOLA-PLC都是基于随机选择的方案自动生成模型，因此会造成大量的组合可能，验证开销大。与Slede比较，EDOLA-PLC中通过允许自定义环境模型，为用户提供了更多选择。

从设计方法学来讲，EDOLA-PLC基于EDOLA语言的三层结构进行设计。与传统的DSL设计方法学相比，EDOLA-PLC的公共模块层实现了多个领域的共性知识，因此易于多领域EDOLA设计时复用。

综上所述，EDOLA-PLC语言，按照EDOLA的领域知识层、公共模块层和验证支持层三层结构进行设计，综合实现了对PLC领域知识表达的方便性、对PLC领域验证需求表达的方便性、在公共知识-实时描述上的易用性和可重用性以及EDOLA-PLC模型的自动验证性。与一般建模语言以及基语言TLA$^+$或其他DSL相比，它实现了领域建模验证语言的设计目标，具有综合的优势。

6.6 本章小结

本章介绍了EDOLA-PLC语言及其工具在码头消防PLC控制系统案例上的建模和验证实践。在建模方面，码头消防案例中的参数、输入输出变量、系统动作、实时约束、待验证属性等可使用EDOLA-PLC语言方便描述；PLC特有的扫描周期模式通过EDOLA-PLC语言的设计得到保证；环境模型无须用户编写，可由EDOLA-PLC编译器自动生成并实现与系统行为描述的组合。建模实践体现了EDOLA-PLC在表达PLC领域特征和实时特征的易用性。在验证方面，码头消防案例的 8 条正确性属性通过EDOLA-PLC的编译器分别基于模型检测和自动定理证明两种方法得到了验证。实验结果说明了验证支持层的转换规则以及面向验证需求的抽象策略在自动验证上的有效性。与已有工作的比较进一步说明了EDOLA-PLC语言的特点以及与已有语言的关系和优势。

第 7 章 结束语

7.1 工作总结

本论文以领域建模验证语言EDOLA为研究对象，提出基于三层结构（领域知识层、公共模块层和验证支持层）的EDOLA设计方法，综合实现了语言的领域特征描述易用性、复用性和自动验证性。论文主要包括以下工作：

1. 介绍了EDOLA语言领域知识层的研究方法及其在生产调度和PLC控制系统两个典型领域的实践。领域知识层的研究按照领域界定、领域知识提取、语言描述和形式语义定义四个步骤进行。在生产调度算法实践中，提取了生产调度问题、时间Petri网模型、算法的五个步骤、顺序执行模式等领域特征并提出两类验证需求，方便了生产调度算法的描述；在PLC控制实践中，提取了变量类别、环境模式、PLC和环境的交互模式等领域特征以及五种常见验证需求，方便了PLC控制软件的描述。定义了领域特征对应的领域操作符并基于TLA$^+$给出领域操作符（以及不能用操作符表达的领域知识）所对应的形式语义。领域知识与验证需求的提取以及操作符的定义为EDOLA的语法设计提供了基础；操作符的形式语义为EDOLA语言的语义解释以及自动验证提供了支持。
2. 以时间特征为例，介绍了EDOLA语言公共模块层的设计方法和实践。通过面向时间特征的知识提取，定义了两类基本时间操作符和四类高级时间操作符，便于描述各种常见时间约束，如动作的持续时间、动作间时间间隔以及延时、最后期限、超时等更高层时间概念。基于TLA$^+$ 定义了各时间操作符的形式语义。操作符的语义定义分为强语义和弱语义两种，语义表达形式上考虑了易于验证的特征。提出了一个TLA$^+$实时模块*RealTimeNew*，以封装时间操作符的语义解释，每一时间操作符被解释为此模块中的一个动作定义。基于模块和动作语义的实时公共模块层研究框架易于扩展新的时间操作符，也可用于其它公共知识的组织。

3. 介绍了EDOLA验证支持层的研究方法和实践。提出了基于中间语言的转换方法，实现多个EDOLA语言之间的验证支持重用。中间语言选用形式语言TLA$^+$的子集subTLA，保证了自动验证性。在EDOLA到subTLA的编译过程中，提出面向验证需求的抽象策略，以减少验证过程中的状态空间和搜索空间。在subTLA的自动定理证明方法中，制定了subTLA关键语法成分：集合、函数等表达式到一阶逻辑的有效编码规则，使得转换后的subTLA模型能够被SPASS等自动定理证明工具直接处理，从而实现了无限空间模型的自动验证。
4. 设计了PLC领域建模验证语言EDOLA-PLC的原型并实现了相关工具。EDOLA-PLC语言的设计综合实现了领域知识层、公共模块层和验证支持层的研究成果；编辑器实现了用户交互接口；编译器实现了语言的语法、语义检查和基于转换的自动验证，方便了EDOLA-PLC语言的应用。
5. 介绍了EDOLA-PLC语言及其工具在码头消防PLC控制系统案例上的建模和验证实践。在建模方面，基于EDOLA-PLC可方便描述码头消防案例中的参数、输入输出变量、系统动作、实时约束和待验证属性； PLC的扫描周期模式通过EDOLA-PLC语言的设计而保证；环境模型可由EDOLA-PLC编译器自动生成并实现其与系统行为描述的组合。建模实践说明了 EDOLA-PLC在表达PLC领域特征和实时特征上的易用性。在验证方面，码头消防案例的 8 条正确性属性通过EDOLA-PLC的编译器分别基于模型检测和自动定理证明两种方法得到了验证。实验结果说明了验证支持层转换规则以及面向验证需求的抽象策略在自动验证上的有效性。与已有工作的比较进一步说明了EDOLA-PLC语言的特点及其与已有语言的关系和优势。

7.2 研究展望

对领域建模验证语言EDOLA的研究和实践，将来还有如下工作要考虑：

1. EDOLA-PLC语言原型的进一步精化，如针对常见设备如阀门、吊杆等，添加更多的领域知识操作符；针对组件交互模式添加多种同步操作符；在系统动作表示方面，区分触发条件和触发效果，采用基于规则（Rule-based）的表达方法，以使语言描述更简洁和易用。

2. EDOLA工具的进一步完善。在EDOLA编辑器方面，我们将考虑实现人机交互的EDOLA模型生成方法，用户只需在人机交互对话框中按提示填入适当内容，系统即可自动生成对应的EDOLA模型文本表示，以进一步增加语言的用户友好性。在EDOLA编译器方面，还需增加对实时验证工具的支持。我们将考虑使用模型检测工具UPPAAL以及自动定理证明工具CVC3对EDOLA语言提供验证支持的研究和实践，尤其要考虑带实数理论的CVC3工具在实时模型验证上的应用实践。
3. EDOLA-PLC语言的进一步应用实践。我们已收集了多个现实的PLC控制应用系统案例。除码头消防PLC控制系统之外，我们将基于EDOLA-PLC语言对一些典型的舞台控制PLC系统（如灵山梵宫舞台、奥运开幕式舞台等）案例的建模和验证实践开展研究。
4. 基于EDOLA-PLC语言的代码自动生成研究。虽然EDOLA的设计是面向验证的，但若能提供EDOLA-PLC到常用PLC编程语言如IL、梯形图等表示的自动转换，将会增强EDOLA语言在工业界的可用性。因此，下一步我们也将考虑EDOLA-PLC模型到IL代码的自动生成研究以及工具支持。
5. 生产调度算法领域建模验证语言EDOLA-SHOP的设计和实践。基于本文的EDOLA设计结构、生产调度领域算法的领域知识提取和表示，我们将考虑生产调度算法领域建模验证语言EDOLA-SHOP的设计和实践，以提高形式验证在此领域的易用性，也是对EDOLA设计框架的进一步检验。

参考文献

[1] Clarke E M, Grumberg O, Peled D. Model Checking. The MIT Press, 1999.

[2] The NuSMV homepage: http://nusmv.irst.itc.it/.

[3] The Spin homepage: http://spinroot.com/spin/whatispin.html.

[4] The UPPAAL homepage: http://www.uppaal.com/.

[5] The Coq homepage: http://Coq.INRIA.fr/.

[6] The Isabelle homepage: http://www.cl.cam.ac.uk/research/hvg/Isabelle/.

[7] Nieuwenhuis R, Oliveras A, Tinelli C. Solving SAT and SAT Modulo Theories: From an abstract Davis–Putnam–Logemann–Loveland procedure to DPLL(T). Journal of ACM, 2006, 53(6):937–977.

[8] The Z3 homepage: http://research.microsoft.com/en-us/um/redmond/projects/z3/.

[9] The CVC3 homepage: http://www.cs.nyu.edu/acsys/cvc3/.

[10] The SPASS homepage: http://www.spass-prover.org/index.html.

[11] Emerson E A, Halpern J Y. “Sometimes” and “not never” revisited: on branching versus linear time temporal logic. J. ACM, 1986, 33(1):151–178.

[12] Taha W. Plenary talk III Domain-specific languages. Proceedings of Computer Engineering & Systems, 2008. ICCES 2008. International Conference on, 2008. xxiii–xxviii.

[13] Bowen J P, Hinchey M G. Ten Commandments of Formal Methods ...Ten Years Later. Computer, 2006, 39(1):40–48.

[14] Hall A. Seven Myths of Formal Methods. IEEE Softw., 1990, 7(5):11–19.

[15] Ghezzi C, Jazayeri M, Mandrioli D. Fundamentals of Software Engineering. Prentice Hall, 2nd edition, 2000.

[16] Alur R, Dill D L. A theory of timed automata. Theoretical Computer Science, 1994, 126(2):183–235.

[17] Peterson J L. Petri Net Theory and the Modeling of Systems. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1981.

[18] Lightfoot D. Formal Specification using Z. Macmillan, 1991.

[19] Abrial J R. The B book - Assigning Programs to Meanings. Cambridge University Press, 1996.

[20] Börger E, Stärk R F. Abstract State Machines—A Method for High-Level System Design and Analysis. Springer-Verlag, 2003.

[21] Budkowski S, Dembinski P. An introduction to Estelle: a specification language for distributed systems. Comput. Netw. ISDN Syst., 1987, 14(1):3–23.

[22] The CASL homepage: http://www.informatik.uni-bremen.de/cofi/wiki/index.php/CASL.

[23] Logrippo L, Faci M, Haj-Hussein M. An introduction to LOTOS: learning by examples. Comput. Netw. ISDN Syst., 1992, 23(5):325–342.

[24] Habrias H, Frappier M. Software Specification Methods. Wiley-ISTE, 2006.

[25] The Promela language: http://www.dai-arc.polito.it/dai-arc/manual/tools/jcat/main/node168.html.

[26] Lamport L. Specifying Systems. Addison-Wesley, 2002.

[27] Chaochen Z, Hansen M R. Duration Calculus: A formal Approach to Real-time systems. Monographs in Theoretical Computer Science, 2004.

[28] Jifeng H, Li X, Liu Z. rCOS: a refinement calculus of object systems. Theoretical Computer Science, 2006, 365(1-2):109–142.

[29] He J. Linking Semantic Models. Proceedings of Proceedings of 4th International Colloquium on Theoretical Aspects of Computing(ICTAC 2007), 2007. 26–28.

[30] Lin H. PAM: A Process Algebra Manipulator. Formal Methods in System Design, 1995, 7(3):243–259.

[31] Lin H. Predicate μ-Calculus for Mobile Ambients. J. Comput. Sci. Technol., 2005, 20(1):95–104.

[32] Kitamura T, Lin H. Specifying Properties for Modular Pi-Calculus. Proceedings of Second IEEE/IFIP International Symposium on Theoretical Aspects of Software Engineering. IEEE Computer Society, 2008. 201–208.

[33] 王戟. 混成实时系统形式化开发方法和技术的研究[D]. 长沙: 国防科技大学, 1995.

[34] Kamin S N, Hyatt D. A special-purpose language for picture-drawing. Proceedings of DSL'97: Proceedings of the Conference on Domain-Specific Languages on Conference on Domain-Specific Languages (DSL), 1997, Berkeley, CA, USA: USENIX Association, 1997. 23–23.

[35] Preuer S. A domain-specific language for industrial automation. Proceedings of Software Engineering (Workshops), volume 106 of *LNI*, 2007. 349–352.

[36] Kumar S, Mandelbaum Y, Yu X, et al. ESP: a language for programmable devices. Proceedings of PLDI '01: Proceedings of the ACM SIGPLAN 2001 conference on Programming language design and implementation, New York, NY, USA: ACM, 2001. 309–320.

[37] Delaval G, Rutten E. A domain-specific language for task handlers generation, applying discrete controller synthesis. Proceedings of SAC '06: Proceedings of the 2006 ACM symposium on Applied computing, New York, NY, USA: ACM, 2006. 901–905.

[38] Chandra S, Richards B, Larus J R. Teapot: A Domain-Specific Language for Writing Cache Coherence Protocols. IEEE Trans. Softw. Eng., 1999, 25(3):317–333.

[39] Atkins D L, Ball T, Bruns G, et al. Mawl: A Domain-Specific Language for Form-Based Services. IEEE Trans. Software Eng., 1999, 25(3):334–346.

[40] Voellmy A, Hudak P. Nettle: A Language for Configuring Routing Networks. Proceedings of Domain-Specific Languages, IFIP TC 2 Working Conference, DSL 2009, Oxford, UK, July 15-17, 2009, Proceedings, volume 5658 of *Lecture Notes in Computer Science*. Springer, 2009. 211–235.

[41] Consel C. Domain-Specific Program Generation; International Seminar, Dagstuhl Castle. Number 3016 in Lecture Notes in Computer Science, State-of-the-Art Survey. Springer-Verlag, 2004: 19–29.

[42] Consel C, Réveillère L. Domain-Specific Program Generation; International Seminar, Dagstuhl Castle. Number 3016 in Lecture Notes in Computer Science, State-of-the-Art Survey. Springer-Verlag, 2004: 165 – 179.

[43] Spinellis D. Notable Design Patterns for Domain Specific Languages. Journal of Systems and Software, 2001, 56(1):91–99.

[44] Deursen A, Klint P, Visser J. Domain-specific languages: an annotated bibliography. ACM SIGPLAN Notices, 2000, 35(6):26–36.

[45] Mernik M, Heering J, Sloane A M. When and how to develop domain-specific languages. ACM Computing Survey, 2005, 37(4):316–344.

[46] Gray J, Fisher K, Consel C, et al. DSLs: the good, the bad, and the ugly. Proceedings of OOPSLA Companion '08: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications, New York, NY, USA: ACM, 2008. 791–794.

[47] The UML homepage: http://www.uml.org/.

[48] Sztipanovits J, Karsai G. Model-Integrated Computing. Computer, 1997, 30(4):110–111.

[49] Porter J, Lattmann Z, Hemingway G, et al. The ESMoL Modeling Language and Tools for Synthesizing and Simulating Real-Time Embedded Systems. Proceedings of 15th IEEE Real-Time and Embedded Technology and Applications Symposium, San Francisco, CA, 2009.

[50] Tolvanen J P. Domain-Specific Modeling and Code Generation for Product Lines. Proceedings of SPLC '06: Proceedings of the 10th International on Software Product Line Conference, Washington, DC, USA: IEEE Computer Society, 2006. 229.

[51] Tolvanen J P, Kelly S. Defining Domain-Specific Modeling Languages to Automate Product Derivation: Collected Experiences. Proceedings of Proceedings of the 9th International Conference on Software Product Lines, SPLC 2005. Springer, 2005. 198–209.

[52] Tolvanen J P. Making model-based code generator work - practical examples. Embedded Systems Europe, 2005, 64(9):38–41.

[53] OMG EAI: http://www.omg.org/spec/EAI/index.htm.

[54] OMG MARTE: http://www.omgmarte.org/.

[55] Risoldi M, Buchs D. A domain specific language and methodology for control systems GUI specification, verification and prototyping. Proceedings of VLHCC '07: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, Washington, DC, USA: IEEE Computer Society, 2007. 179–182.

[56] Bodeveix J P, Filali M, Lawall J, et al. Formal Methods Meet Domain Specific Languages. In: J Romijn G S, Pol J, (eds.). Proceedings of Fifth International Conference on Integrated Formal Methods, volume 3771 of *Lecture Notes in Computer Science*, Eindhoven, The Netherlands, 2005. 187–206.

[57] Bodeveix J P, Filali M, Lawall J, et al. Applying the B formal method to the Bossa domain-specific language. Proceedings of The 17th Nordic Workshop on Programming Theory (NWPT'05), Copenhagen, Denmark, 2005. 35–38.

[58] Latry F, Mercadal J, Consel C. Staging Telephony Service Creation: A Language Approach. Proceedings of in Principles, Systems and Applications of IP Telecommunications, IPTComm, New-York, NY, USA: ACM Press, 2007.

[59] 张乃孝, 琚小明, 孙猛. Garden: 一种面向领域语言的集成开发环境. 中国科学（E辑：信息科学）, 2008, 38(12):2084–2098.

[60] 范少峰. 面向语言的领域语言开发方法研究[D]. 北京: 北京大学, 2006.

[61] 陈斌. GarAda系统的设计与实现[M]. 北京: 北京大学, 2001.

[62] 陈旭盛. 领域语言VERT的设计及变换型实现[M]. 北京: 北京大学, 2001.

[63] 张迎春, 黄林鹏. upDSL :一种描述动态更新策略的领域特定语言. 微电子学与计算机, 2008, 25(10):34–39.

[64] 宋柱梅, 迪. 基于MIC理论构建嵌入式装备控制领域建模语言的研究. 制造业自动化, 2005, 27(12):46–49.

[65] 周艳明. 基于领域专用语言的应用软件自动生成. 计算机工程与应用, 2003, (10):124–127.

[66] 王成, 孟晨. 面向自动测试系统领域语言的设计与实现. 军械工程学院学报, 2007, 19(5):42–45.

[67] 李英军, 刘鸿儒. SEIS++ : 一个油气勘探领域软件建造和集成的模式语言. 计算机学报, 2000, 23(1):102–107.

[68] 王爱菊. 面向银行信贷业务领域语言的研究和应用[M]. 北京: 哈尔滨工程大学, 2007.

[69] Abadi M, Lamport L, Merz S. A TLA solution to the RPC-Memory specification problem. Proceedings of Formal Systems Specification: The RPC-Memory Specification Case Study. SpringerVerlag, 1996. 21–66.

[70] Merz S. TLA$^+$ Case Study: A Resource Allocator. Rapport de recherche, LORIA, August, 2004. http://www.loria.fr/publications/2004/A04-R-101/A04-R-101.ps.

[71] Grov G, Michaelson G, Ireland A. Formal verification of concurrent scheduling strategies using TLA. Proceedings of ICPADS '07: Proceedings of the 13th International Conference on Parallel and Distributed Systems, 2007. 1–6.

[72] Lamport L. The temporal logic of actions. ACM Transactions on Programming Languages and Systems, 1994, 16(3):872– 923.

[73] Lewis R. Programming industrial control systems using IEC 1131-3,volume 50 of Control Engineering Series. Stevenage, United Kingdom: The Institution of Electrical Engineers, 1998.

[74] Bonfatti F, Monari P, Sampieri U. IEC 1131-3 Programming Methodology. Fontaine, France: CJ International, 1999.

[75] Rausch M, Krogh B H. Formal verification of PLC programs. Proceedings of Proceedings of American Control Conference 1998., volume 1, 1998. 234–238.

[76] Mertke T, Frey G. Formal Verification of PLC-programs generated from Signal Interpreted Petri Nets. Proceedings of Proceedings of the SMC 2001, Tucson (AZ) USA, 2001. 2700–2705.

[77] Jiménez-Fraustro F, Rutten E. A Synchronous Model of IEC 61131 PLC Languages in SIGNAL. Proceedings of ECRTS '01: Proceedings of the 13th Euromicro Conference on Real-Time Systems, Washington, DC, USA: IEEE Computer Society, 2001. 135.

[78] RWang, XSong, Gu M. Modelling and verification of program logic controllers using timed automata. IET Software, 2007, 4:127–131.

[79] Wan H, Chen G, Song X, et al. Formalization and Verification of PLC Timers in Coq. Computer Software and Applications Conference, Annual International, 2009, 1:315–323.

[80] Zhou M, He F, Gu M, et al. Translation-Based Model Checking for PLC Programs. Computer Software and Applications Conference, Annual International, 2009, 1:553–562.

[81] Zhang H, Merz S, Gu M. Specifying and Verifying PLC Systems with TLA+. Theoretical Aspects of Software Engineering, Joint IEEE/IFIP Symposium on, 2009, 0:293–294.

[82] Pressman R S. Software Engineering : A Practitioner's Approach (4th Edition). McGraw-Hill, 1997.

[83] Simos M A. Organization domain modeling (ODM): formalizing the core domain modeling life cycle. SIGSOFT Softw. Eng. Notes, 1995, 20(SI):196–205.

[84] Kang K C, Cohen S G, Hess J A, et al. Feature-oriented domain analysis (FODA) feasibility study. Technical report, Software Engineering Institute, Carnegie Mellon Univeristy, 1990.

[85] Taylor R N, Tracz W, Coglianese L. Software development using domain-specific software architectures. SIGSOFT Softw. Eng. Notes, 1995, 20(5):27–38.

[86] Coplien J, Hoffman D, Weiss D. Commonality and Variability in Software Engineering. IEEE Softw., 1998, 15(6):37–45.

[87] Pnueli A. The Temporal Logic of Programs. Proceedings of Proceedings of 18th Annual Symposium on Foundations of Computer Science, 1977. 46–57.

[88] Gabbay D M, Pnueli A, Shelah S, et al. On the Temporal Basis of Fairness. Proceedings of POPL, 1980. 163–173.

[89] Dwyer M B, Avrunin G S, Corbett J C. Property specification patterns for finite-state verification. Proceedings of FMSP '98: Proceedings of the second workshop on Formal methods in software practice, New York, NY, USA: ACM, 1998. 7–15.

[90] Dwyer M B, Avrunin G S, Corbett J C. Patterns in property specifications for finite-state verification. Proceedings of ICSE '99: Proceedings of the 21st international conference on Software engineering, New York, NY, USA: ACM, 1999. 411–420.

[91] Corbett J C, Dwyer M B, Hatcliff J, et al. A Language Framework for Expressing Checkable Properties of Dynamic Software. Proceedings of Proceedings of the 7th International SPIN Workshop on SPIN Model Checking and Software Verification, London, UK: Springer-Verlag, 2000. 205–223.

[92] Yi Y, Wang D W. Soft computing for scheduling with batch setup times and earliness-tardiness penalties on parallel machines. Journal of Intelligent Manufacturing, 2003, 14(3):311–322.

[93] Kim D W, Kim K H, Jang W, et al. Unrelated parallel machine scheduling with setup times using simulated annealing. Robotics and Computer-Integrated Manufacturing, 2002, 18(3-4):223–231.

[94] Hillion H P, Proth J M. Using timed Petri nets for the scheduling of job-shop systems. Engineering Costs and Production Economics, 1989, 17(1-4):149–154.

[95] Wang L C, Wu S Y. Modeling with colored timed object-oriented Petri nets for automated manufacturing systems. Computers and Industrial Engineering, 1998, 34(2):463–480.

[96] Christian A, Francois R. A Petri net model and a general method for on and off-line multi-resource shop floor scheduling with setup times. International Journal of Production Economics, 2001, 74(1-3):63.

[97] Zhang W, Freiheit T, Yang H. Dynamic scheduling in flexible assembly system based on timed Petri nets model. Robotics and Computer-Integrated Manufacturing, 2005, 21(6):550–558.

[98] Kim Y W, Suzuki T, Narikiyo T. FMS scheduling based on timed Petri Net model and reactive graph search. Applied Mathematical Modelling, 2007, 31(6):955–970.

[99] R L Graham J K L, Kan A H G R. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics, 1979, 5:287–326.

[100] Bowden F D. A brief survey and synthesis of the roles of time in Petri nets. Mathematical and Computer Modelling, 2000, 31(10-12):55–68.

[101] Murata T. Petri Nets: Properties, Analysis and Applications. Proceedings of Proceedings of the IEEE, 1989. 541–580.

[102] 何衍庆. 常用PLC应用手册. 电子工业出版社, 2008.

[103] 胡学林. 可编程控制器原理与应用. 电子工业出版社, 2007.

[104] Manna Z, Pnueli A. A hierarchy of temporal properties (invited paper, 1989). Proceedings of PODC '90: Proceedings of the ninth annual ACM symposium on Principles of distributed computing, New York, NY, USA: ACM, 1990. 377–410.

[105] Singhal A. Real Time Systems: A Survey, 1996.

[106] Lee E A. Embedded Software. Proceedings of Advances in Computers. Academic Press, 2002.

[107] Schneider S. Concurrent and Real-time Systems: the CSP Approach. John Wiley & Sons, 2000.

[108] Léonard L, Leduc G. An introduction to ET-LOTOS for the description of time-sensitive systems. Comput. Netw. ISDN Syst., 1997, 29(3):271–292.

[109] Merlin P, Farber D. Recoverability of Communication Protocols–Implications of a Theoretical Study. IEEE Transactions on Communications, 1976, 24(9):1036–1043.

[110] Mahony B, Dong J S. Timed Communicating Object Z. IEEE Transactions on Software Engineering, 2000, 26(2):150–177.

[111] Konrad S, Cheng B H C. Real-time specification patterns. Proceedings of ICSE '05: Proceedings of the 27th international conference on Software engineering, New York, NY, USA: ACM, 2005. 372–381.

[112] Dong J S, Hao P, Qin S, et al. Timed Automata Patterns. IEEE Transactions on Software Engineering, 2008, 34(6):844–859.

[113] Abadi M, Lamport L. An old-fashioned recipe for real time. ACM Transactions on Programming Languages and Systems., 1994, 16(5):1543–1571.

[114] Leβke F, Merz S. Steam boiler control specification problem: A TLA solution. Proceedings of Formal Methods for Industrial Applications, Specifying and Programming the Steam Boiler Control, London, UK: Springer-Verlag, 1996. 339–358.

[115] Liu Z, Joseph M. Verification of Fault-Tolerance and Real-Time. Proceedings of Proceedings of the 26th Annual International Symposium on Fault-Tolerant Computing. IEEE Computer Society, 1996. 220–229.

[116] Regnier P, Lima G, Andrade A. A TLA+ Formal Specification and Verification of a New Real-Time Communication Protocol. Electronic Notes in Theoretical Computer Science, 2009, 240:221–238.

[117] Boyer M, Roux O H. On the Compared Expressiveness of Arc, Place and Transition Time Petri Nets. Fundam. Inf., 2008, 88(3):225–249.

[118] Clarke E M, Grumberg O, Long D E. Model checking and abstraction. ACM Trans. Program. Lang. Syst., 1994, 16(5):1512–1542.

[119] Tiwari A. Abstraction Based Theorem Proving: An example from the theory of Reals. In: Tinelli C, Ranise S, (eds.). Proceedings of Proceedings of the CADE-19 WOrkshop on Pragmatics of Decision Procedures in Automated Deduction, PDPAR 2003. INRIA, Nancy, 2003. 40–52.

[120] Berendsen J, Vaandrager F. Compositional Abstraction in Real-Time Model Checking. Proceedings of FORMATS '08: Proceedings of the 6th international conference on Formal Modeling and Analysis of Timed Systems, Berlin, Heidelberg: Springer-Verlag, 2008. 233–249.

[121] Henzinger T, Manna Z, Pnueli A. An interleaving model for real time. Proceedings of JCIT: Proceedings of the fifth Jerusalem conference on Information technology, Los Alamitos, CA, USA: IEEE Computer Society Press, 1990. 717–730.

[122] Yu Y, Manolios P, Lamport L. Model checking TLA+ Specifications. In: Pierre L, Kropf T, (eds.). Proceedings of Proceedings of Correct Hardware Design and Verification Methods (CHARME'99), volume 1703 of *Lecture Notes in Computer Science*, Bad Herrenalb, Germany: Springer Verlag, 1999. 54–66.

[123] Abadi M, Lamport L. Open systems in TLA. Proceedings of PODC '94: Proceedings of the thirteenth annual ACM symposium on Principles of distributed computing, New York, NY, USA: ACM, 1994. 81–90.

[124] The JavaCC web site: https://javacc.dev.java.net/.

[125] Lamport L. Real-Time Model Checking is Really Simple. Proceedings of Correct Hardware Design and Verification Methods (CHARME 2005). Springer-Verlag Lecture Notes in Computer Scienc, 2005. 162–175.

[126] Lamport L, Matthews J, Tuttle M, et al. Specifying and verifying systems with TLAp. Proceedings of Proceedings of the 10th workshop on ACM SIGOPS European workshop, 2002.

[127] Angelika M. A Classification of PLC Models and Applications. Proceedings of Proceedings of the 5th Workshop on Discrete Event Systems (WODES 2000), 2000. 239–247.

[128] Heiner M, Menzel T. A Petri net Semantics for the PLC Language Instruction List. Proceedings of Proceedings of IEE Workshop on Discrete Event System (WODES'98), 1998. 161–165.

[129] Canet G, Couffin S, Lesage J J, et al. Towards the automatic verification of PLC programs written in Instruction List. Proceedings of Proceedings of IEEE International conference on Systems, Man and Cybernetics (SMC'2000), 2000. 2449–2454.

[130] HXWillems. Compact Timed Automata for PLC Programs. Technical report, November 24, 1999.

[131] Hanna Y, Rajan H. Slede: Framework for Automatic Verification of Sensor Network Security Protocol Implementations. Proceedings of ICSE '09: 31st International Conference on Software Engineering, 2009.

[132] Hanna Y, Rajan H, Zhang W. Slede: a domain-specific verification framework for sensor network security protocol implementations. Proceedings of WiSec '08: Proceedings of the first ACM conference on Wireless network security, New York, NY, USA: ACM, 2008. 109–118.

致　谢

衷心感谢导师孙家广教授和副导师顾明教授对本人的精心指导和关爱。他们的言传身教将使我终生受益。

特别感谢美国波特兰州立大学宋晓宇教授对本人研究课题以及论文写作方面的指导和帮助。法国INRIA研究院的Stephan Merz在本人访问期间给予了热心的指导，不胜感激。感谢美国微软公司的Leslie Lamport在TLA$^+$语言细节方面给予的耐心解答和帮助。法国INRIA研究院的Charles Consel和Joseph Sifakis与本人在领域语言上的讨论，对本人的工作带来了很多启发和帮助，在此深表谢意。

感谢贺飞老师、万海、王瑞、赵先朋等同窗好友以及Cereus组的每一位成员，在我攻读学位期间，他们在学习和生活上带来很多很多快乐、帮助和关心。

最后感谢我的家人尤其是我的父母，对于我的决定和固执，你们总是理解、支持并做好我的坚强后盾。你们的爱是我努力的动力和源泉。

声　明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人享有著作权的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。

签　名：_____ 日　期：_____

个人简历、在学期间发表的学术论文与研究成果

个人简历

1982 年 6 月 12 日出生于河南省民权县。

1997 年 9 月考入吉林大学计算机系，2001 年 7 月获得工学学士学位。2001 年 7 月免试进入吉林大学计算机系，2004 年 7 月获得硕士学位。

2004 年 9 月考入清华大学计算机系攻读博士学位至今。

发表的学术论文

[1] Hehua Zhang, Ming Gu, Xiaoyu Song. A dead-lock free scheduling with sequence-dependent setup times. International Journal of Advanced Manufacturing Technology, v45, p593-692, November, 2009. (SCI，影响因子 0.743)

[2] Hehua Zhang, Stephan Merz, Ming Gu. Specifying and Verifying PLC systems with TLA+. Proceedings of the 3rd IEEE International Symposium on Theoretical Aspects of Software Engineering (TASE 2009) ,Tianjin, China, pp.293-294, July 29-31, 2009. (EI)

[3] Hehua Zhang, Ming Gu. Modeling job shop scheduling with batches and setup times by timed Petri nets. Mathematical and Computer Modeling, v49, p286-294, January, 2009. (SCI，影响因子 1.032)

[4] Hehua Zhang, Ming Gu and Xiaoyu Song. Modeling and Analysis of Stage Machinery Control Systems by Timed Colored Petri Nets. Proceedings of the 3rd International Symposium on Industrial Embedded Systems, (SIES 2008), Montpellier, France, June 11-13, p103-110, 2008. (EI)

[5] Hehua Zhang, Ming Gu, Xiaoyu Song. Modeling and Analysis of Real-life Job Shop Scheduling Problems by Petri nets. Proceedings of the 41st Annual Simulation Symposium (ANSS 2008), Ottwa, Canada, April 14-16, p279-285, 2008. (EI)

[6] Hehua Zhang, Zhi Guo etc. Software process validation based on model checking. Journal of Computational Information Systems, v1, n3, p577-583, September, 2005. (EI)

参与的科研项目及取得的科研成果

[1] 2006.01- 2009.01: 国家973计划“现代设计大型应用软件的共性基础”子课题六“协同设计大型应用软件体系结构及形式化机理研究” (No.2004CB719406)

[2] 2007.10- 2008.5: 国家863重点项目“面向离散制造的可配置MES产品及行业解决方案” (No.2007AA040701-1)

[3] 2008.01-至今: 国家自然科学基金重点项目“基于定理证明的可信嵌入式软件建模与验证平台研究”（No. 90718039）

[4] 顾明等（译）. 交互式定理证明与程序开发-Coq归纳构造演算的艺术.清华大学出版社，北京，2010.